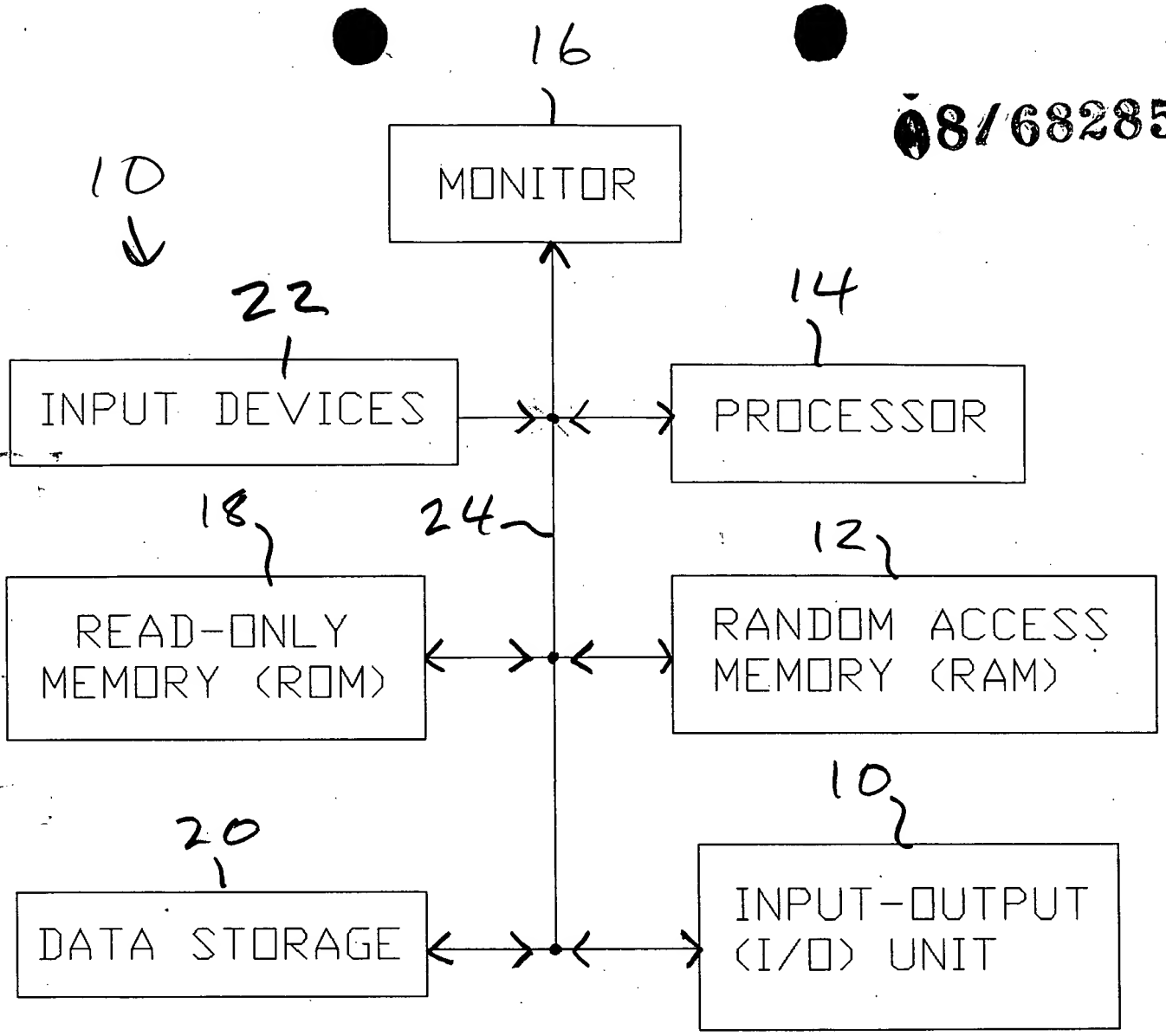

16
MONITOR
10
22
INPUT DEVICES
14
PROCESSOR
18
24
12
READ-ONLY MEMORY (ROM)
RANDOM ACCESS MEMORY (RAM)
10
20
DATA STORAGE
INPUT-OUTPUT (I/O) UNIT

Fig. 1

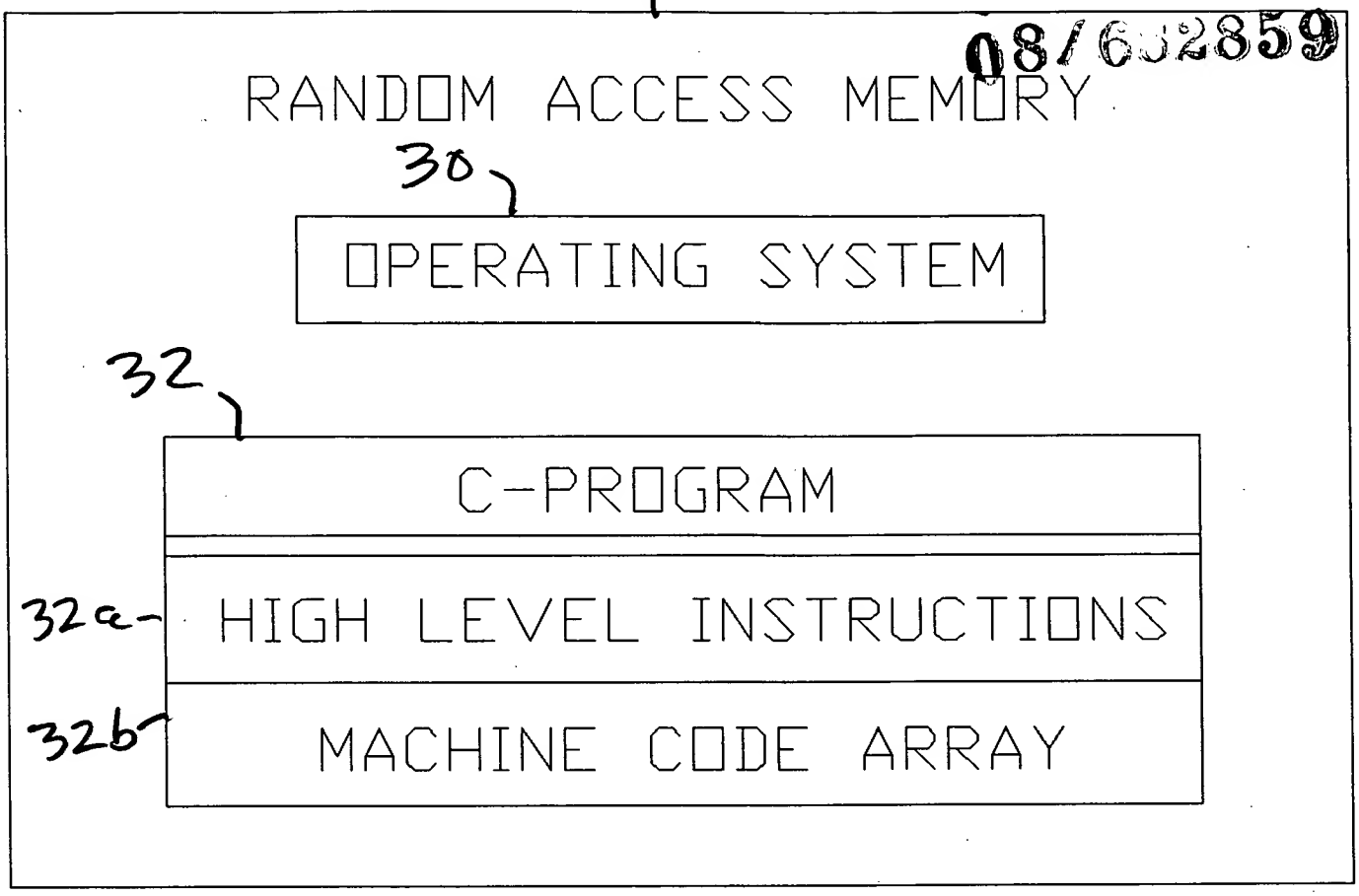


Fig. 2

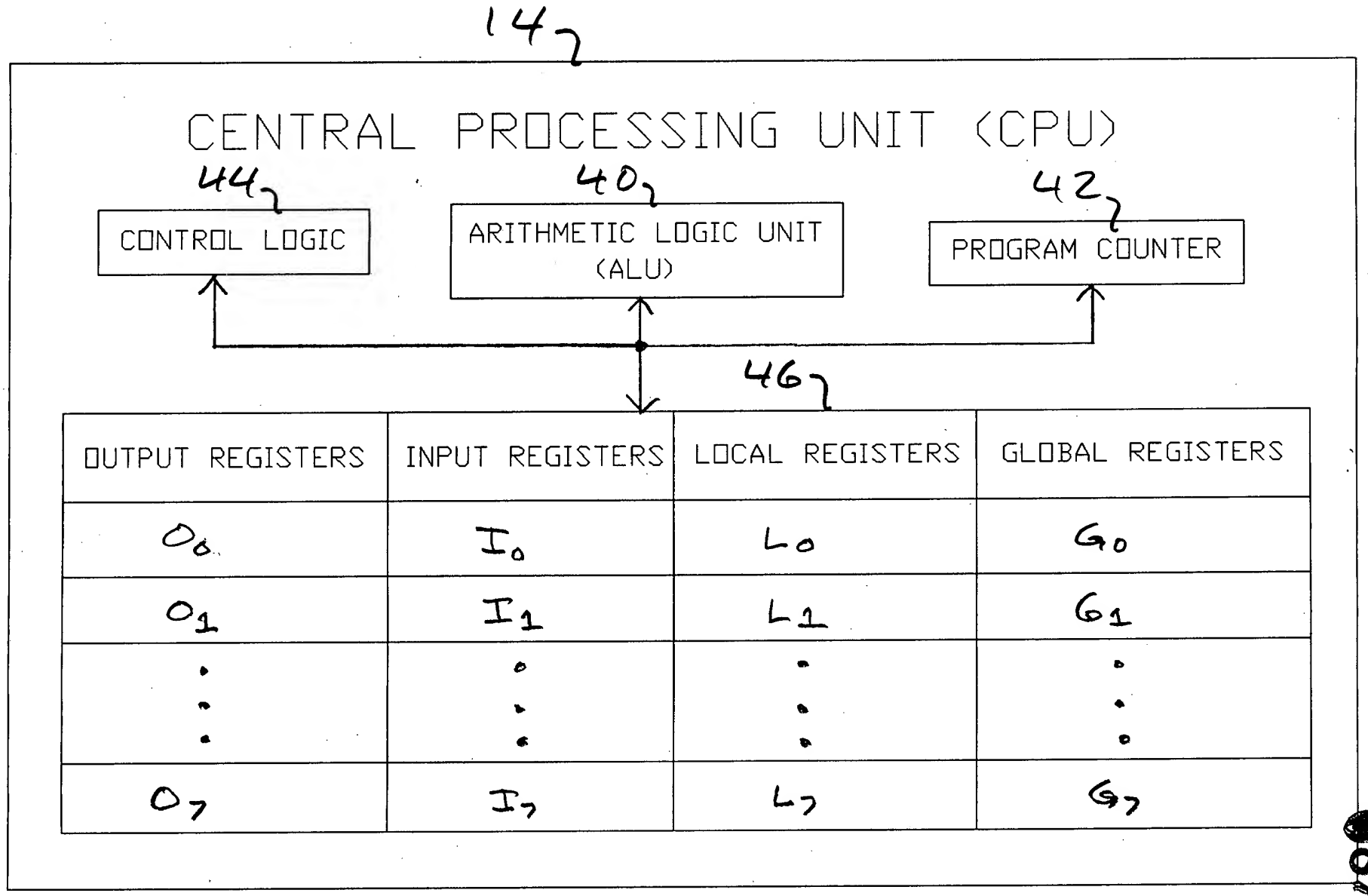


Fig. 3

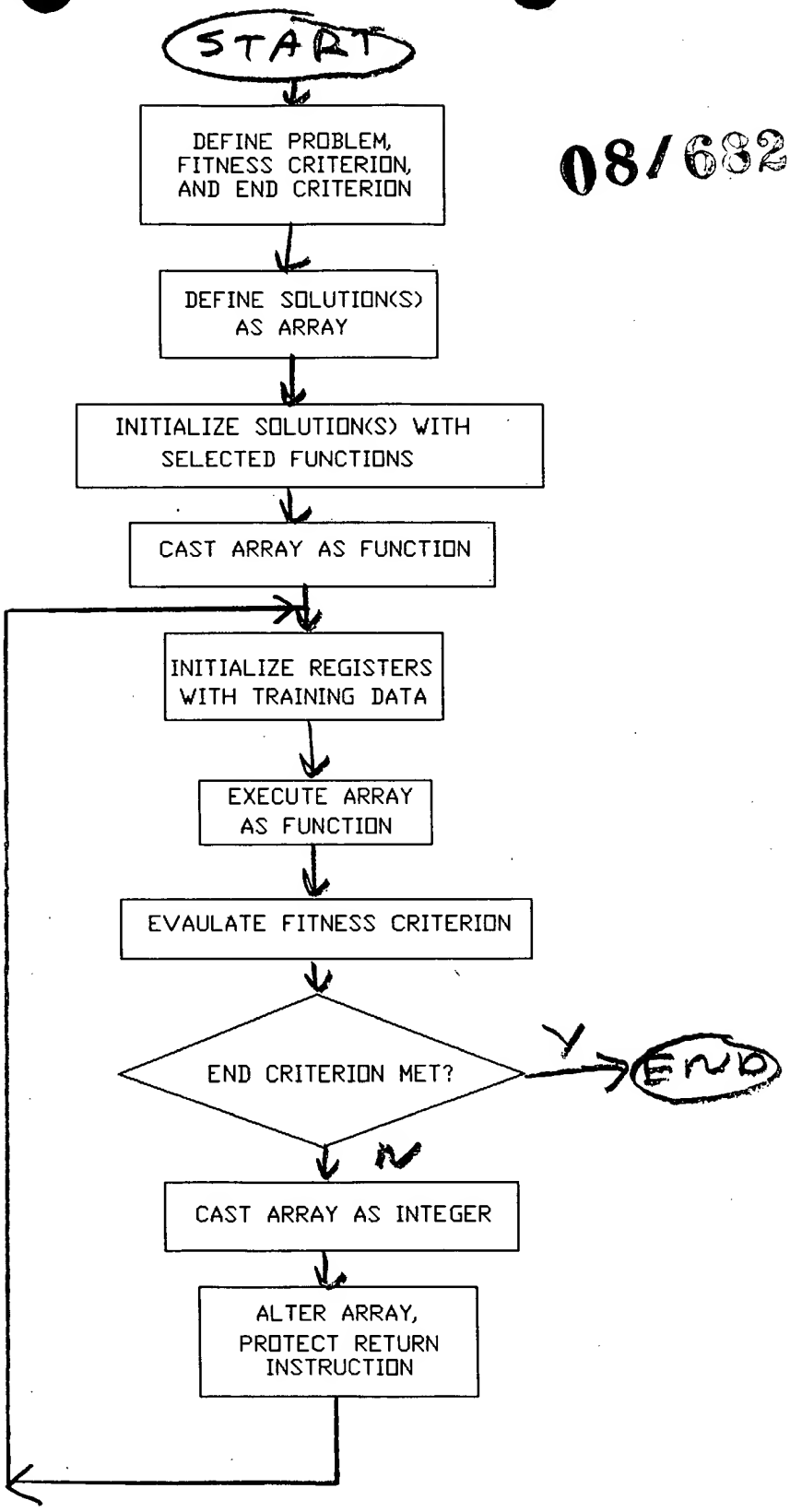

START
DEFINE PROBLEM, FITNESS CRITERION, AND END CRITERION
DEFINE SOLUTION(S) AS ARRAY
INITIALIZE SOLUTION(S) WITH SELECTED FUNCTIONS
CAST ARRAY AS FUNCTION
INITIALIZE REGISTERS WITH TRAINING DATA
EXECUTE ARRAY AS FUNCTION
EVAULATE FITNESS CRITERION
END CRITERION MET?
Y
END
N
CAST ARRAY AS INTEGER
ALTER ARRAY, PROTECT RETURN INSTRUCTION

Fig. 4

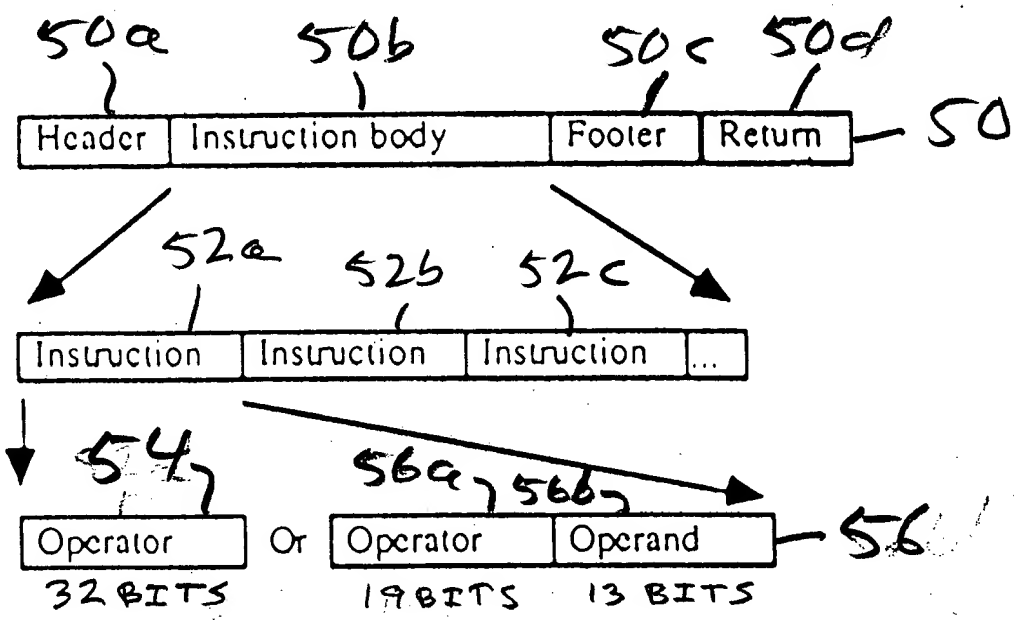

50a
50b
50c
50d
Header
Instruction body
Footer
Return
50
52a
52b
52c
Instruction
Instruction
Instruction
...
54
56a
56b
Operator
Or
Operator
Operand
56
32 BITS
19 BITS
13 BITS

Fig. 5

08/682859

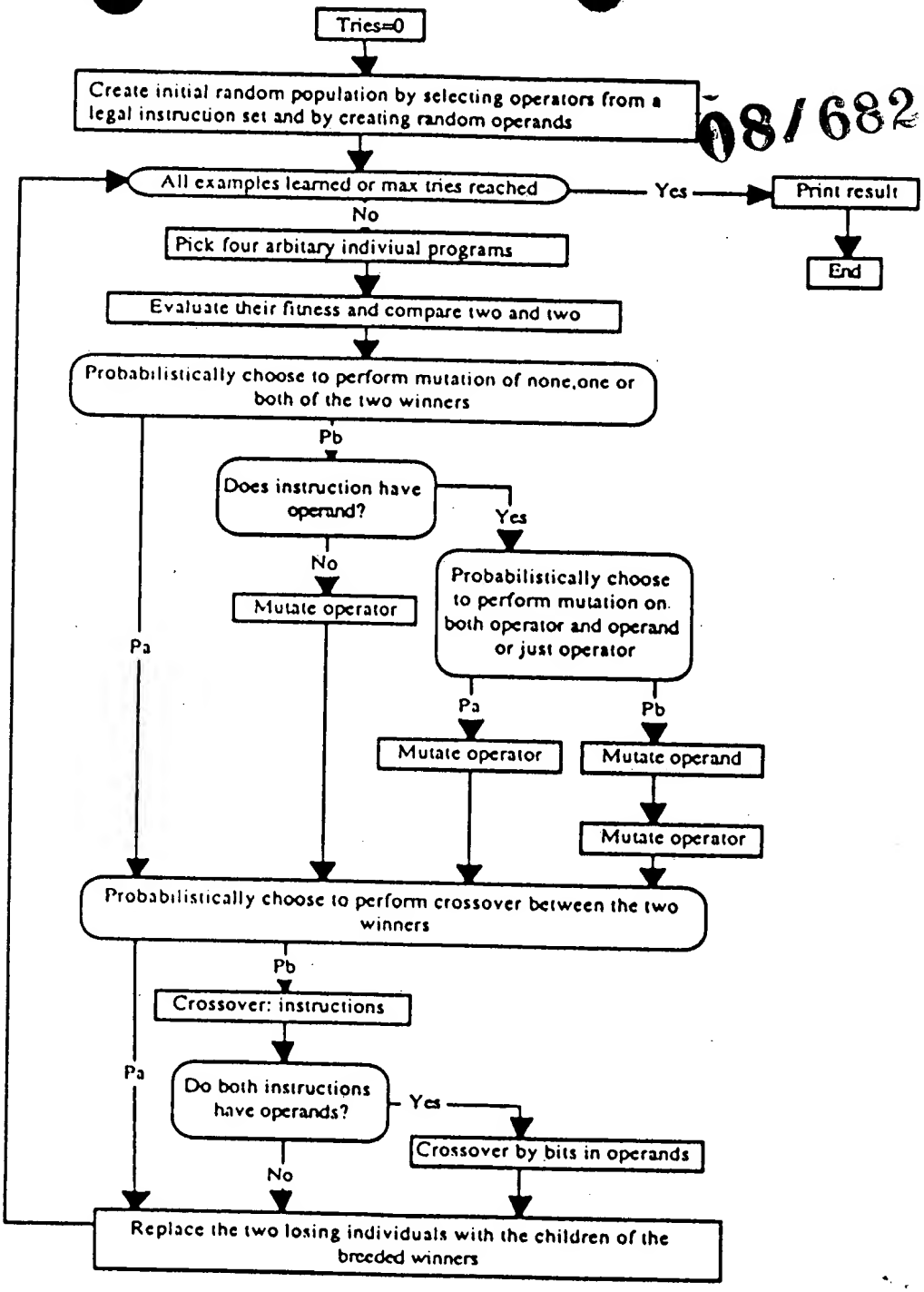


Fig. 6

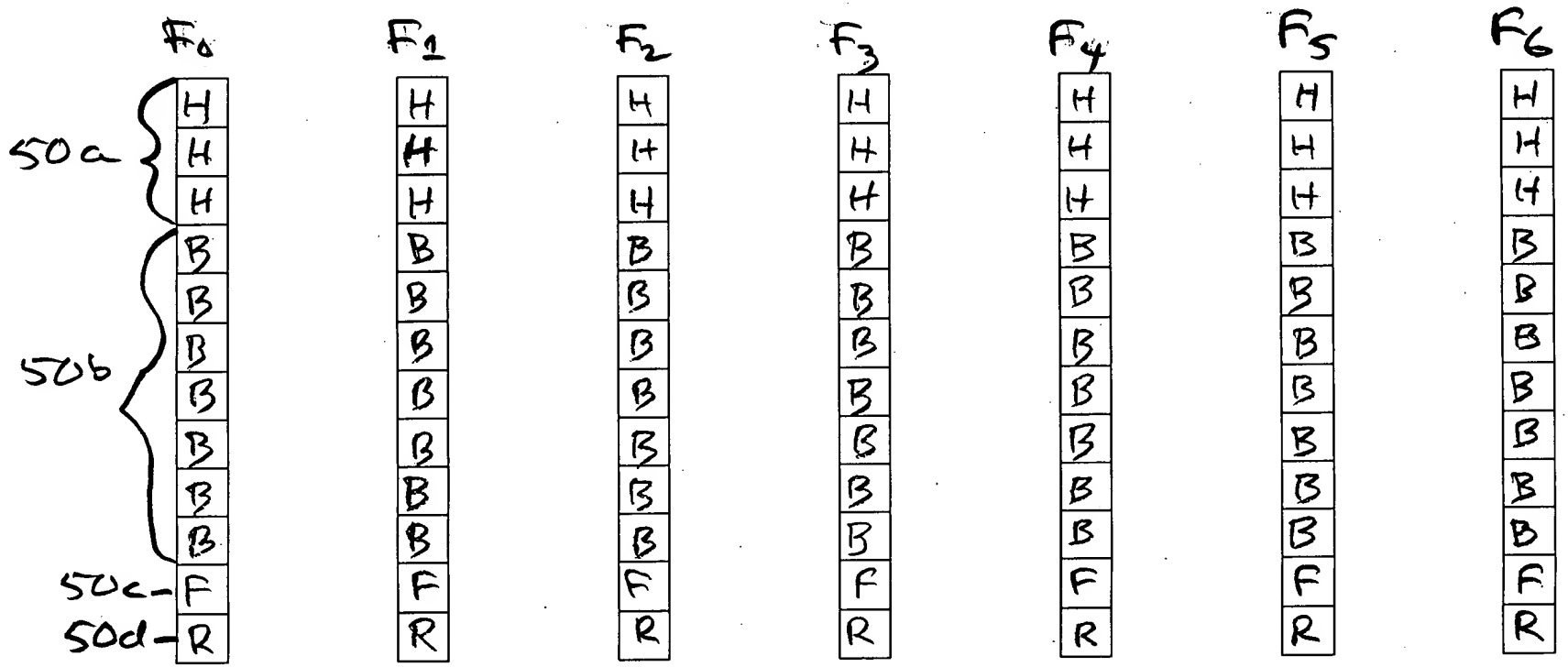


Fig. 7

Fig. 8

08/682859

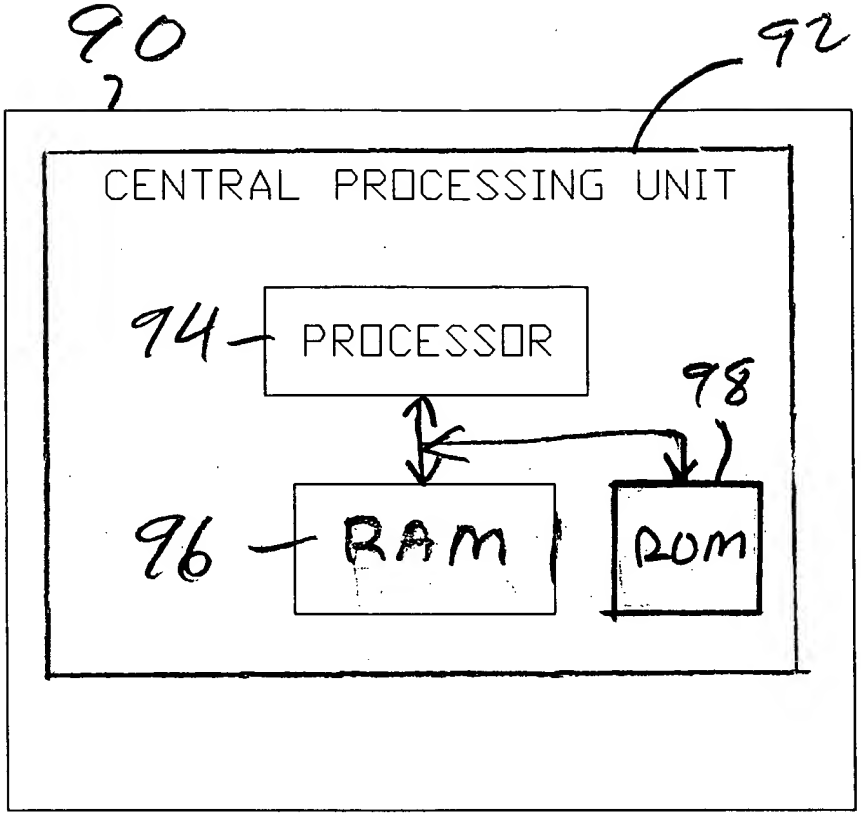


Fig. 9

08/682859

PARENTS

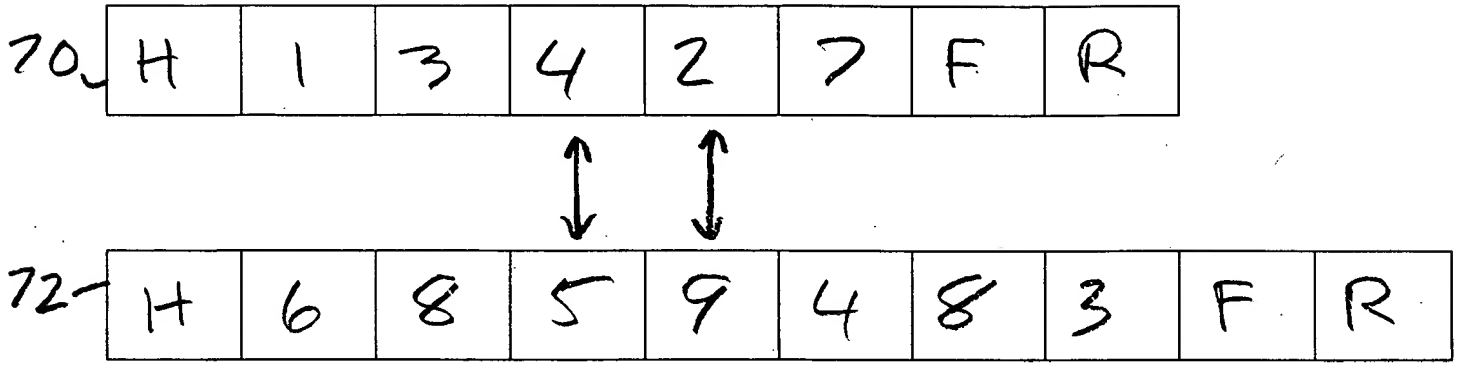


Fig. 10a

CHILDREN

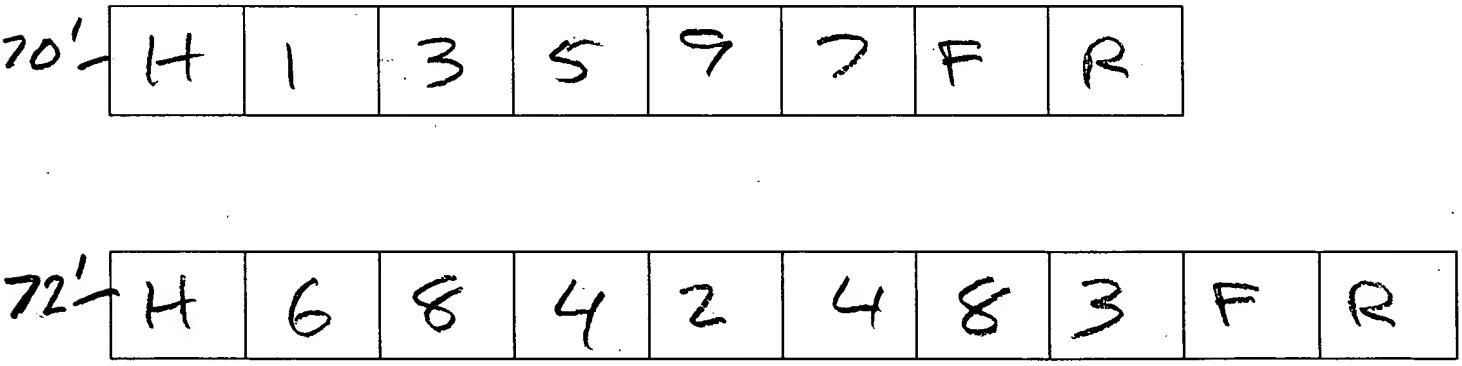


Fig. 10b

PARENTS

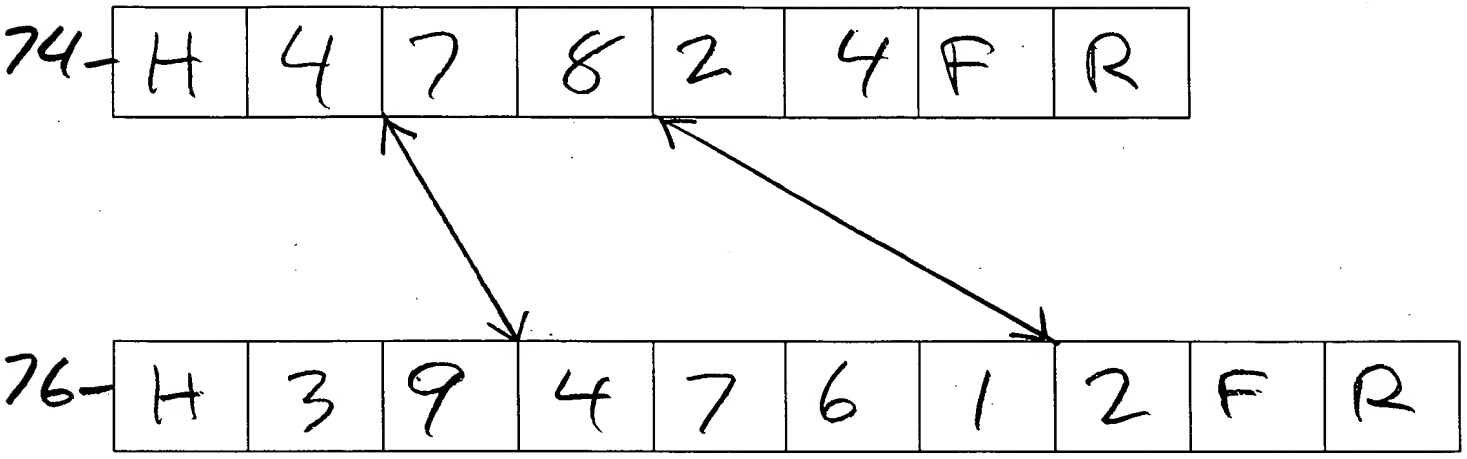


Fig. 11a

CHILDREN

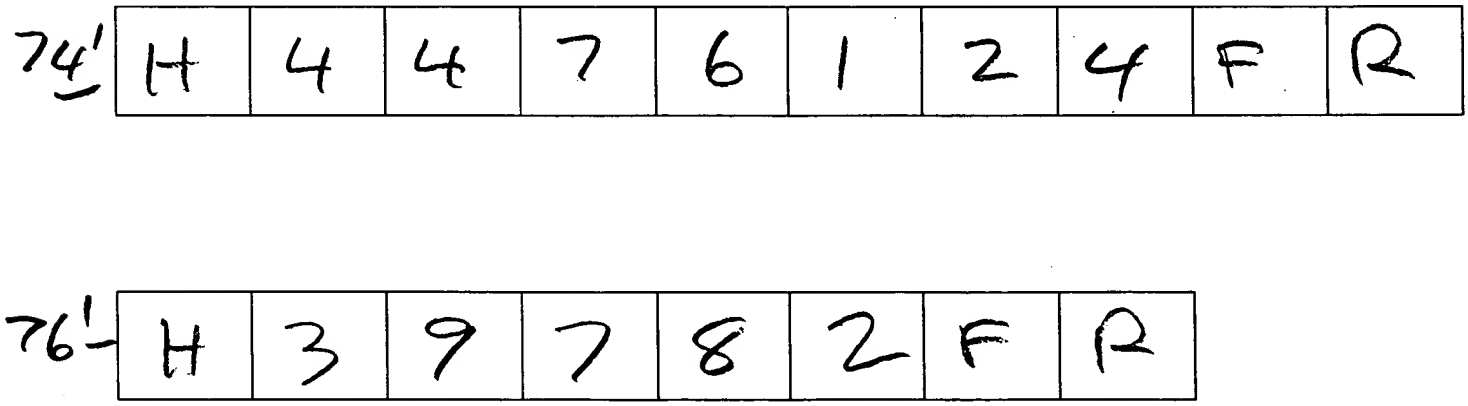


Fig. 11b

PARENTS

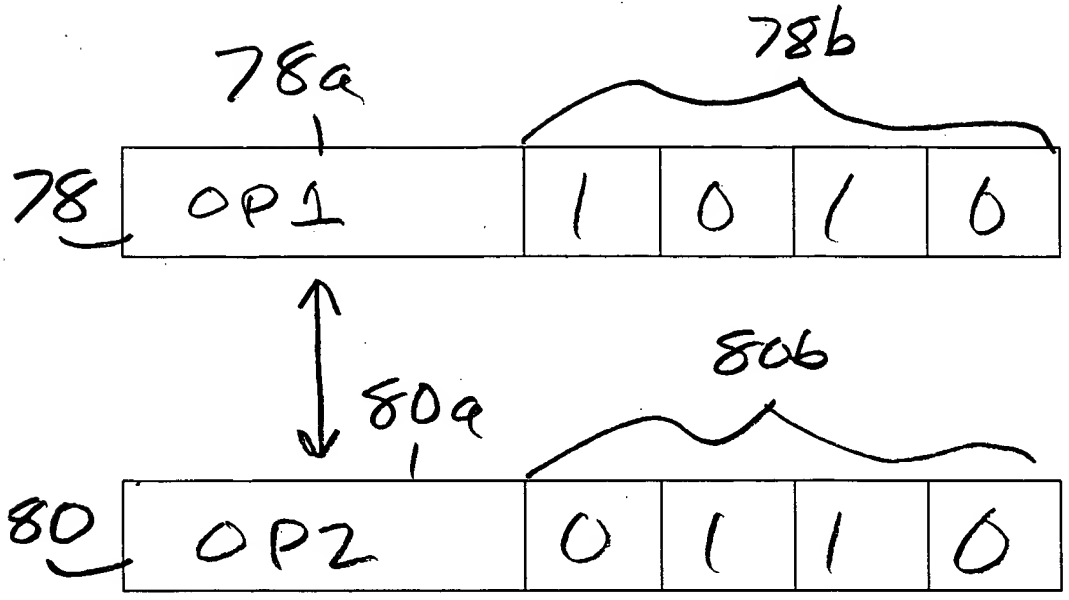


Fig. 12a

CHILDREN

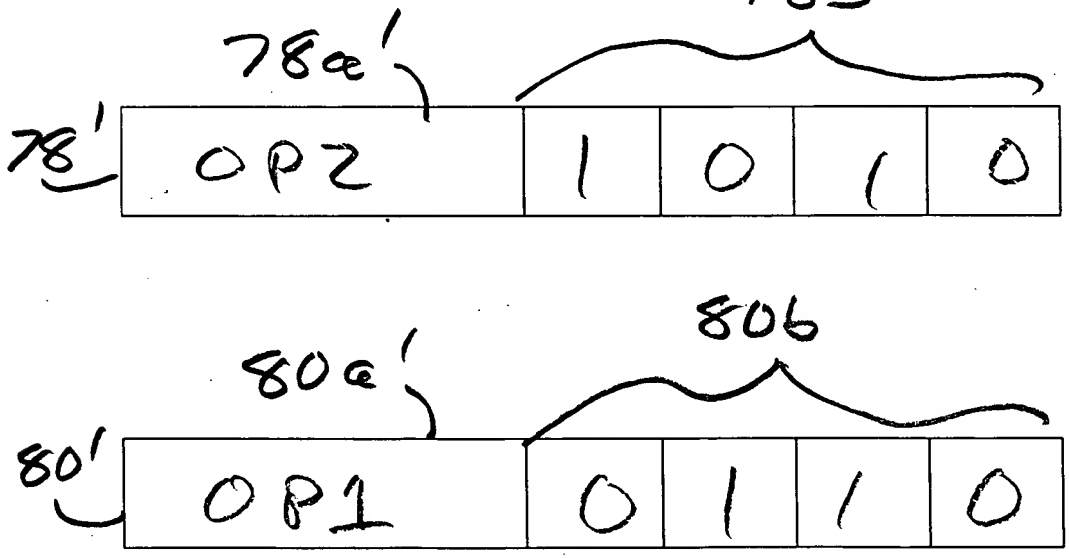


Fig. 12b

PARENTS

08/682859

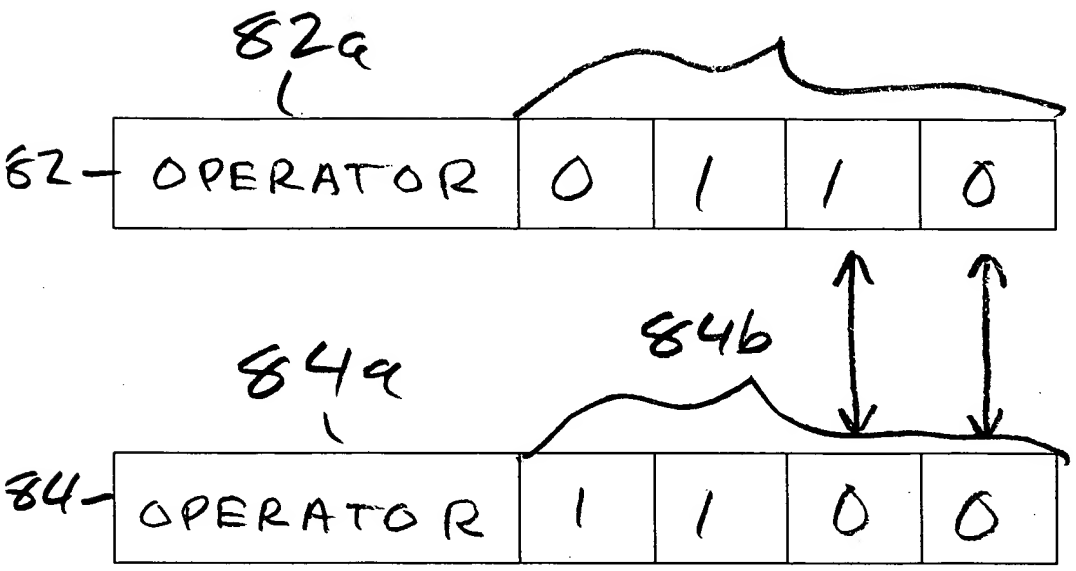


Fig. 13a

CHILDREN

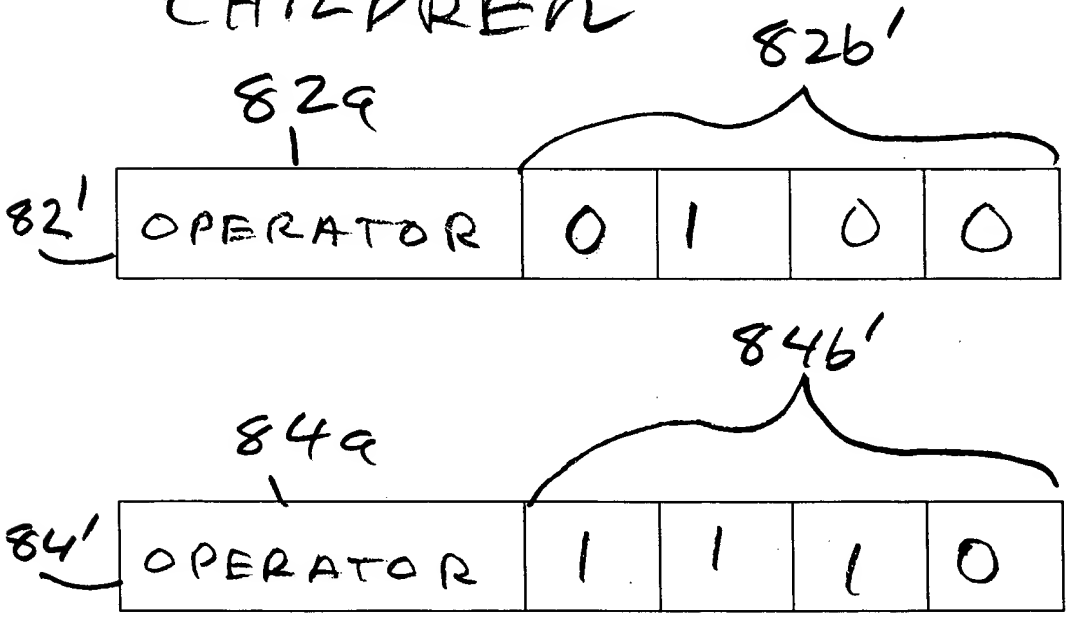


Fig. 13b

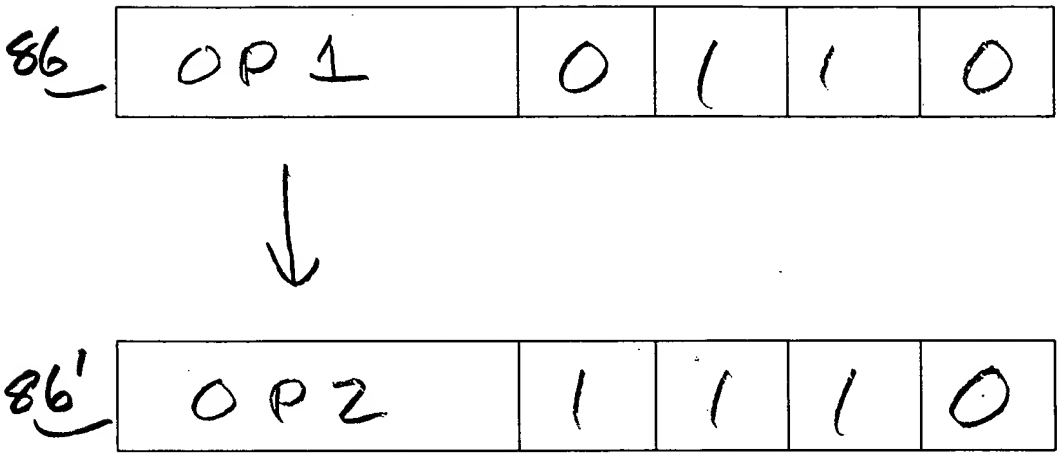

86
OP 1
0
1
1
0
86'
OP 2
1
1
1
0

Fig. 14

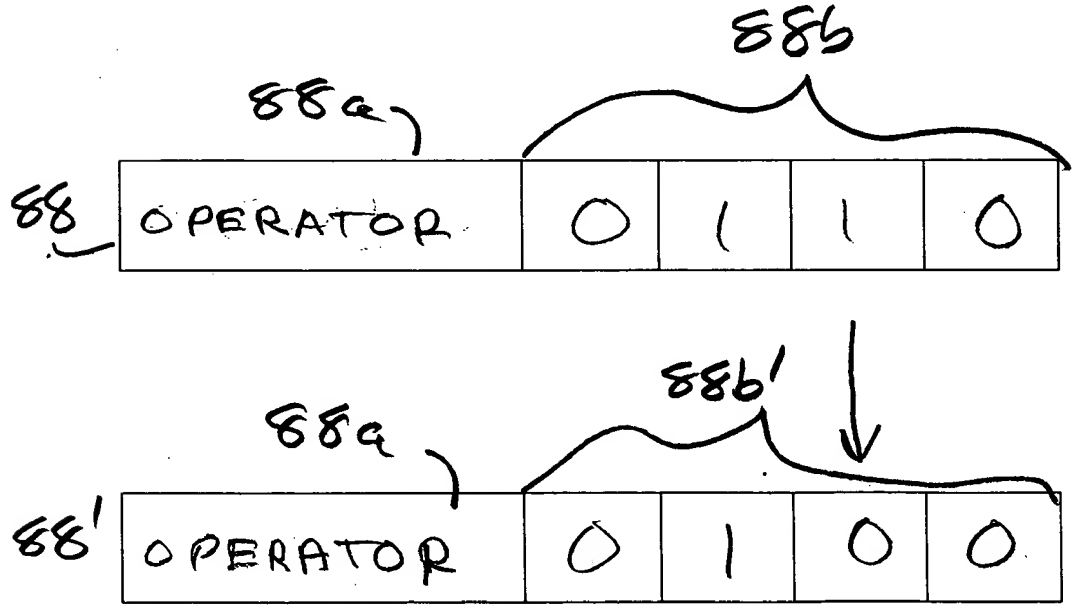

88b
88a
88
OPERATOR
0
1
1
0
88b'
88a
88'
OPERATOR
0
1
0
0

Fig. 15

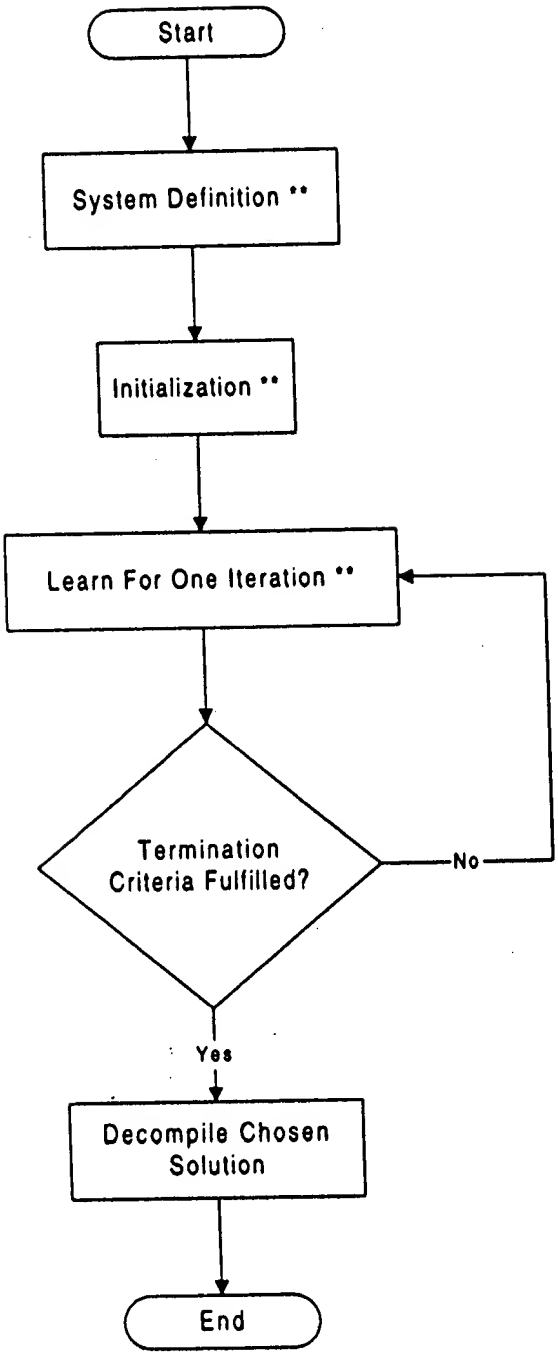


Figure 16a

General Operation of the System

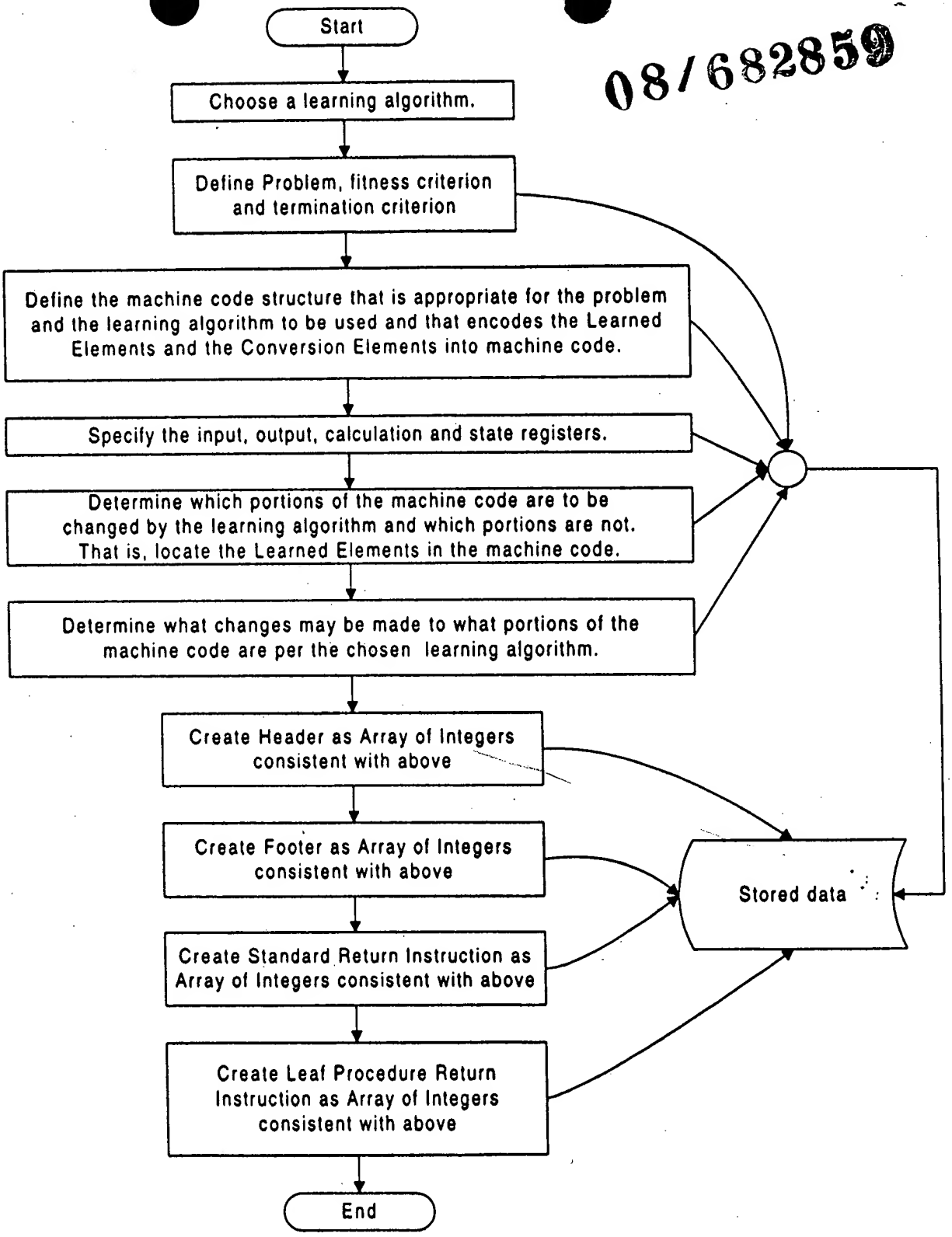


Figure 16b

Details of System Definition

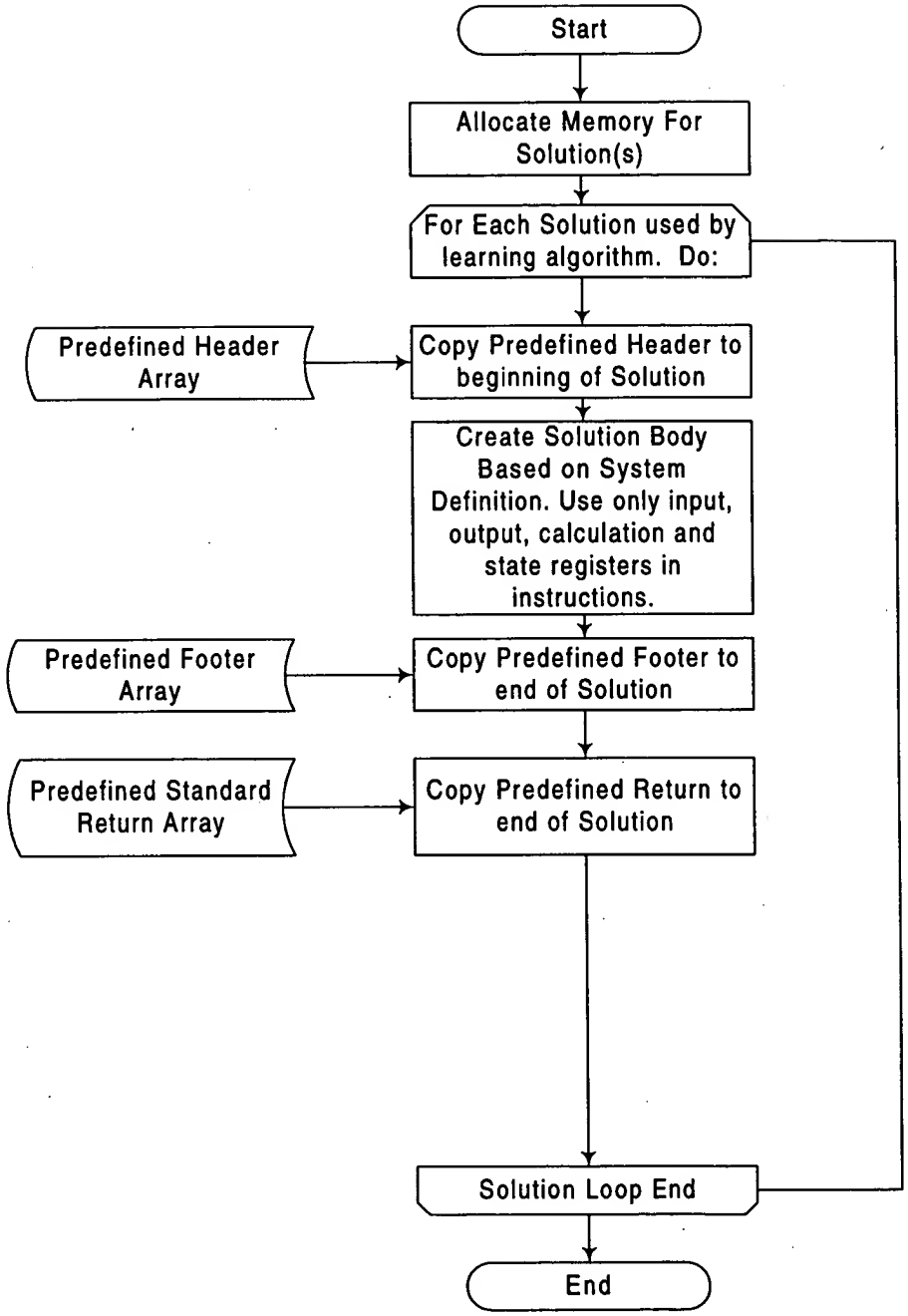
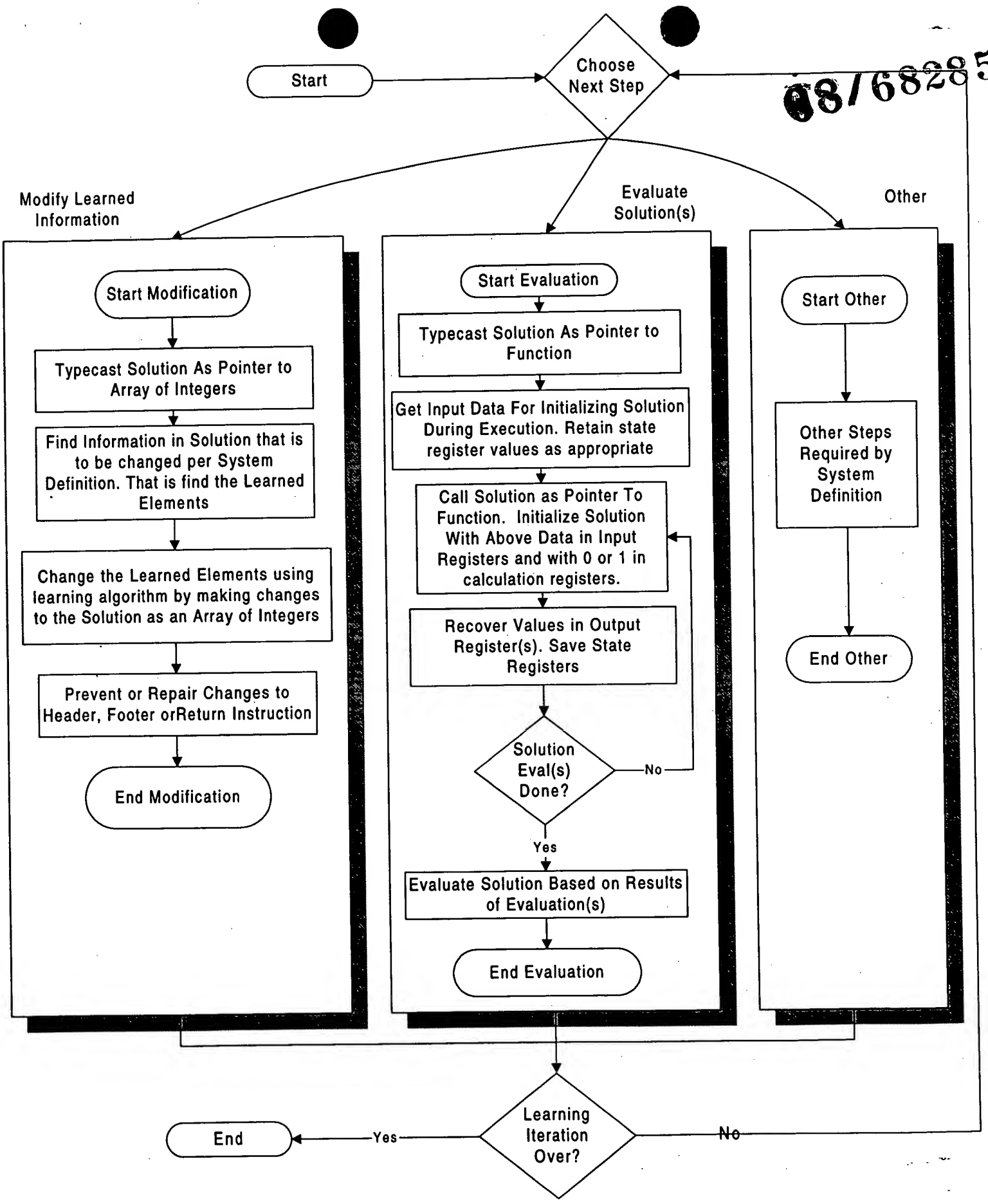


Figure 16c

Initialization

"Solution" means entity or individual



"Solution" means entity or individual

Figure 16d

Learn for One Iteration

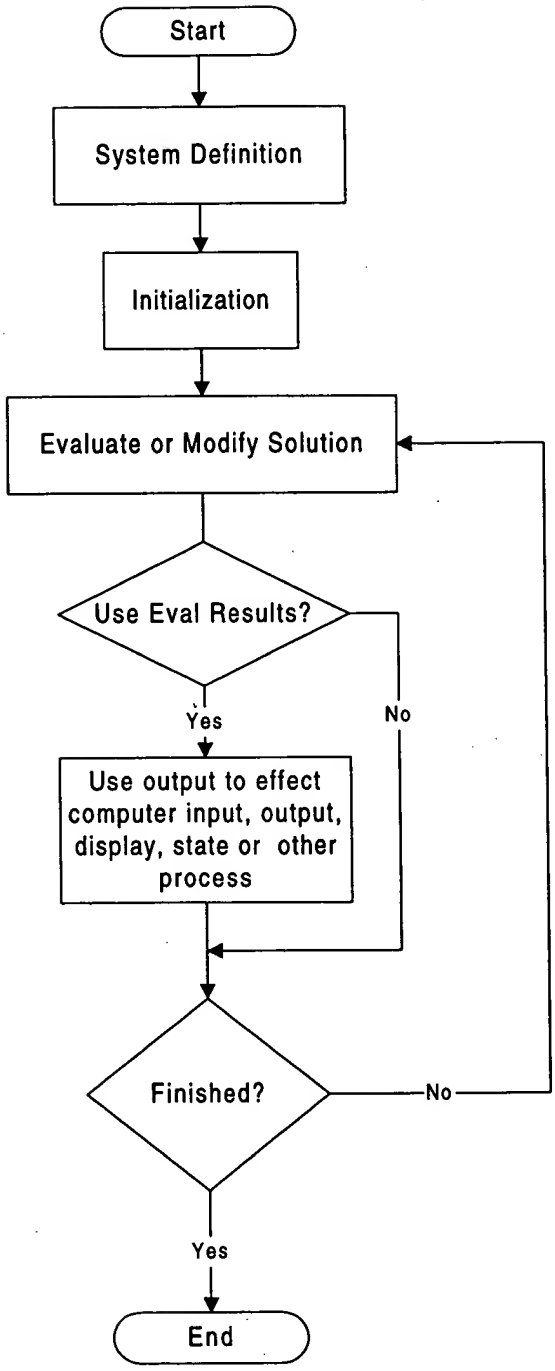


Figure 17a

General Operation of the Invention As Repetitive Computation System

"Solution" means entity or individual

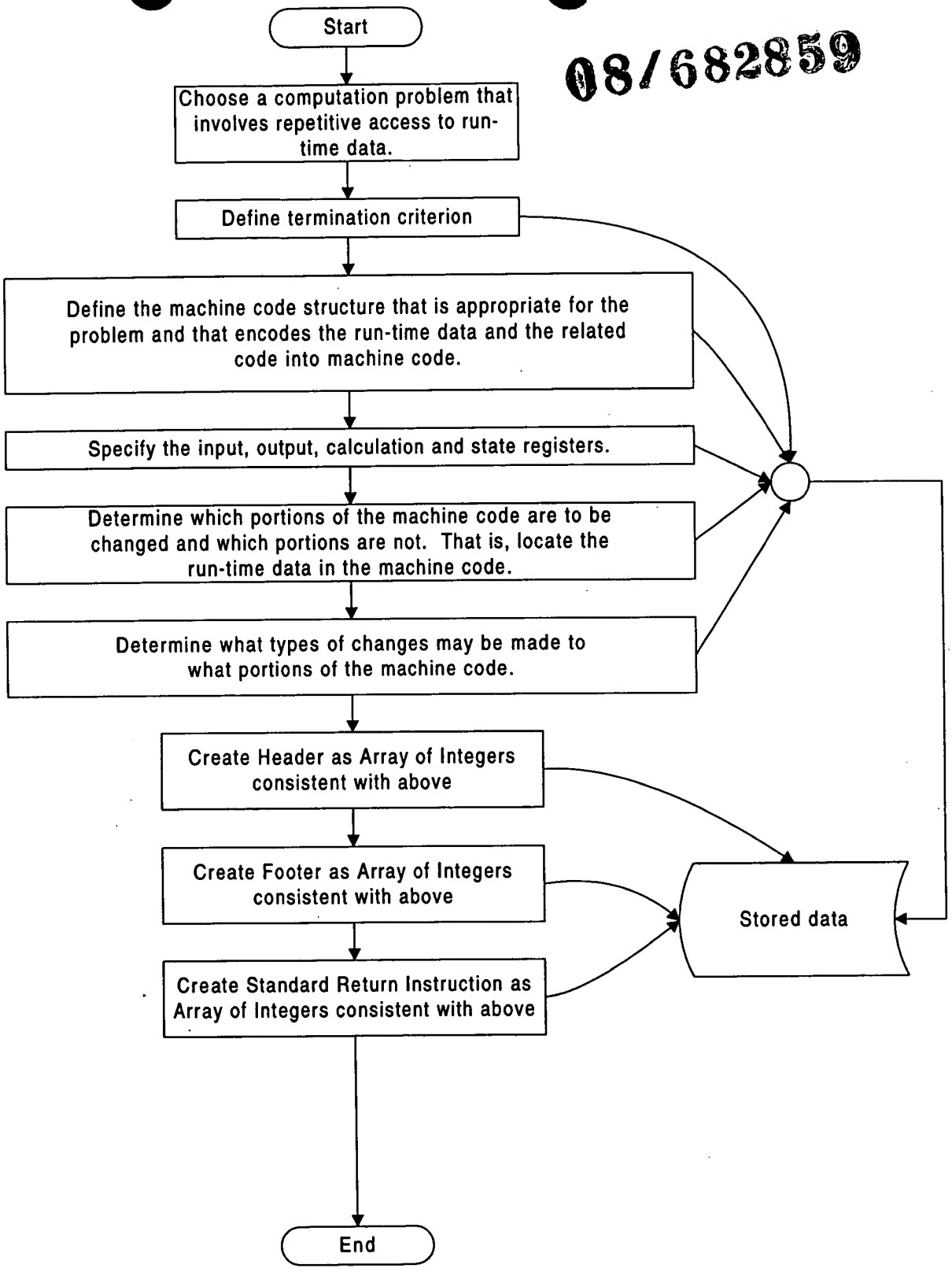


Figure 17b

System Definition

"Solution" means entity or individual

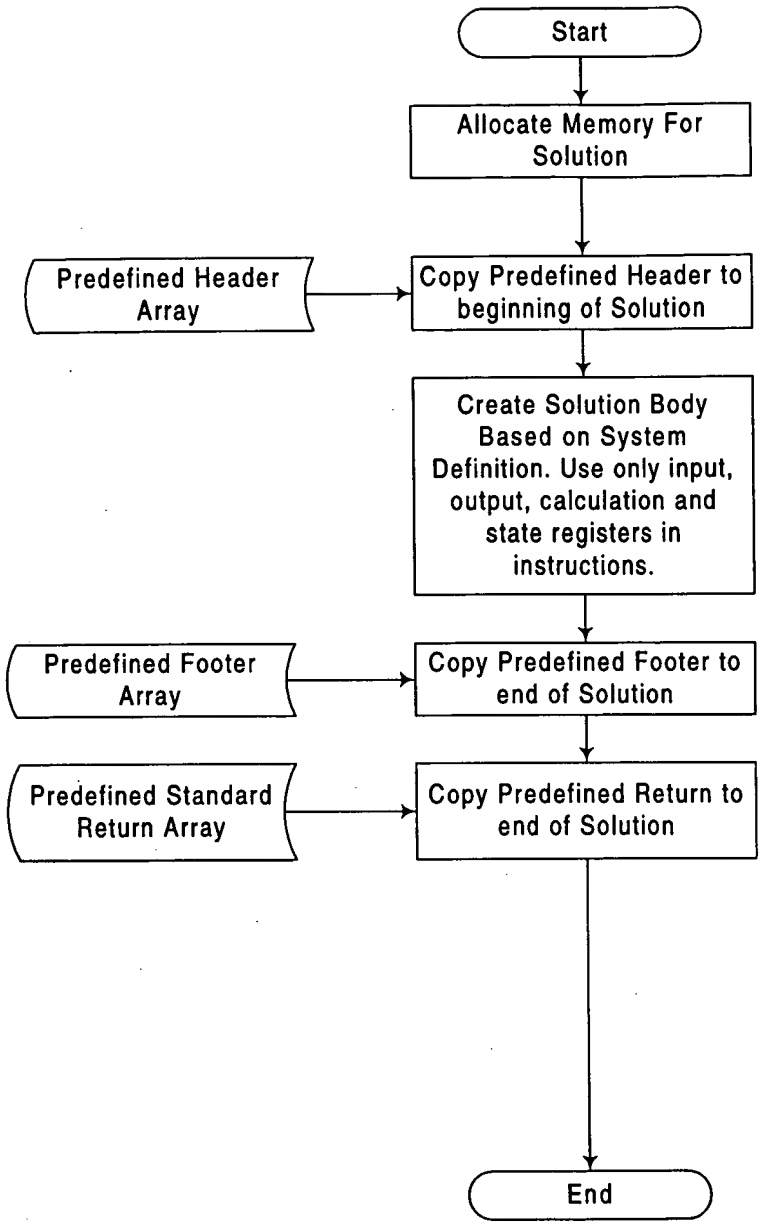
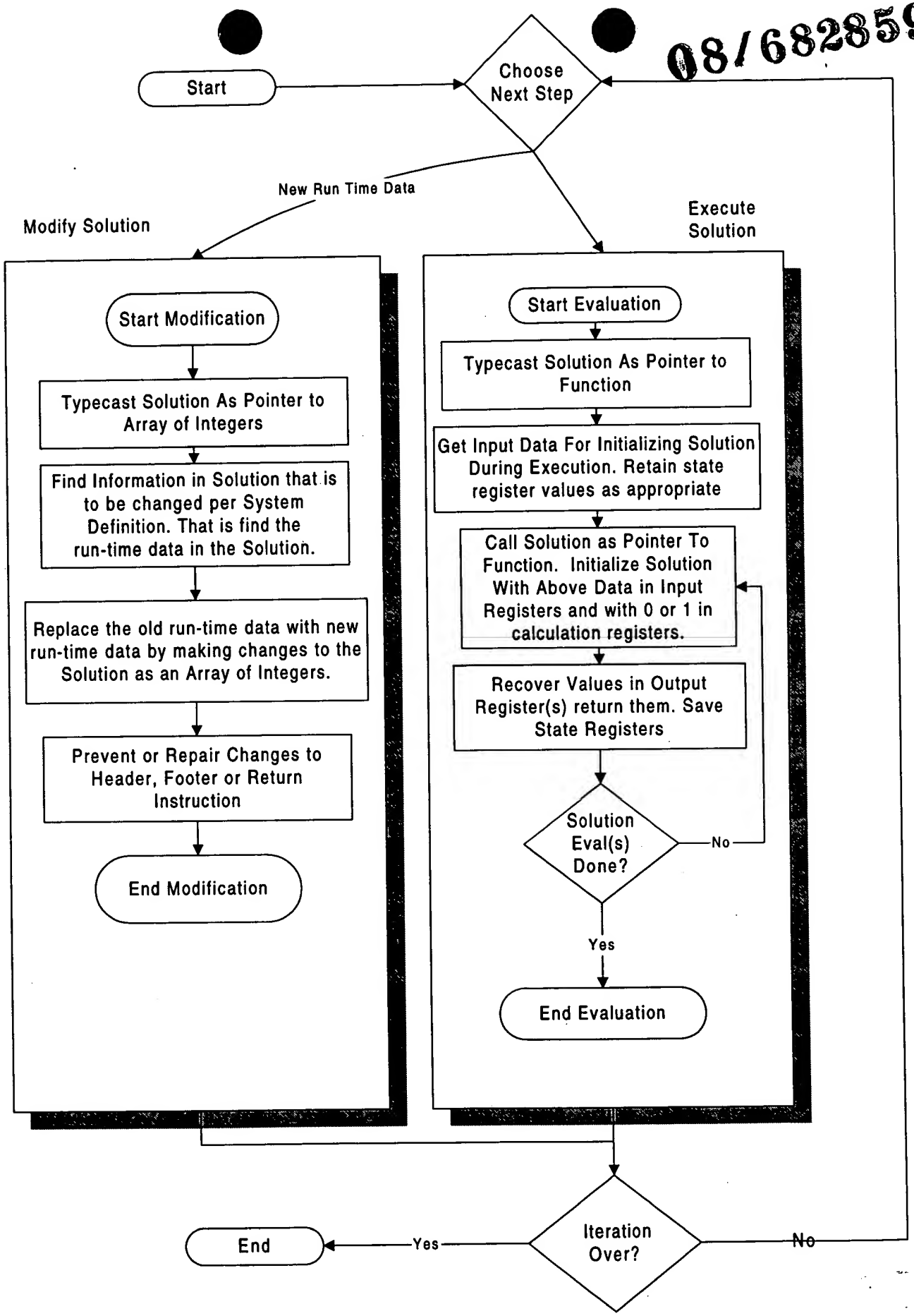


Figure 17c

Initialization

"Solution" means entity or individual



"Solution" means entity or individual

Figure 17d

Evaluate or Modify Function

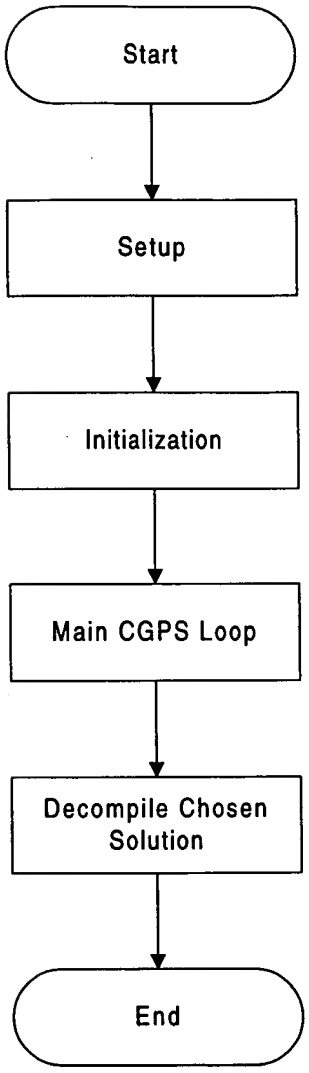

Start
Setup
Initialization
Main CGPS Loop
Decompile Chosen Solution
End

Figure 18a

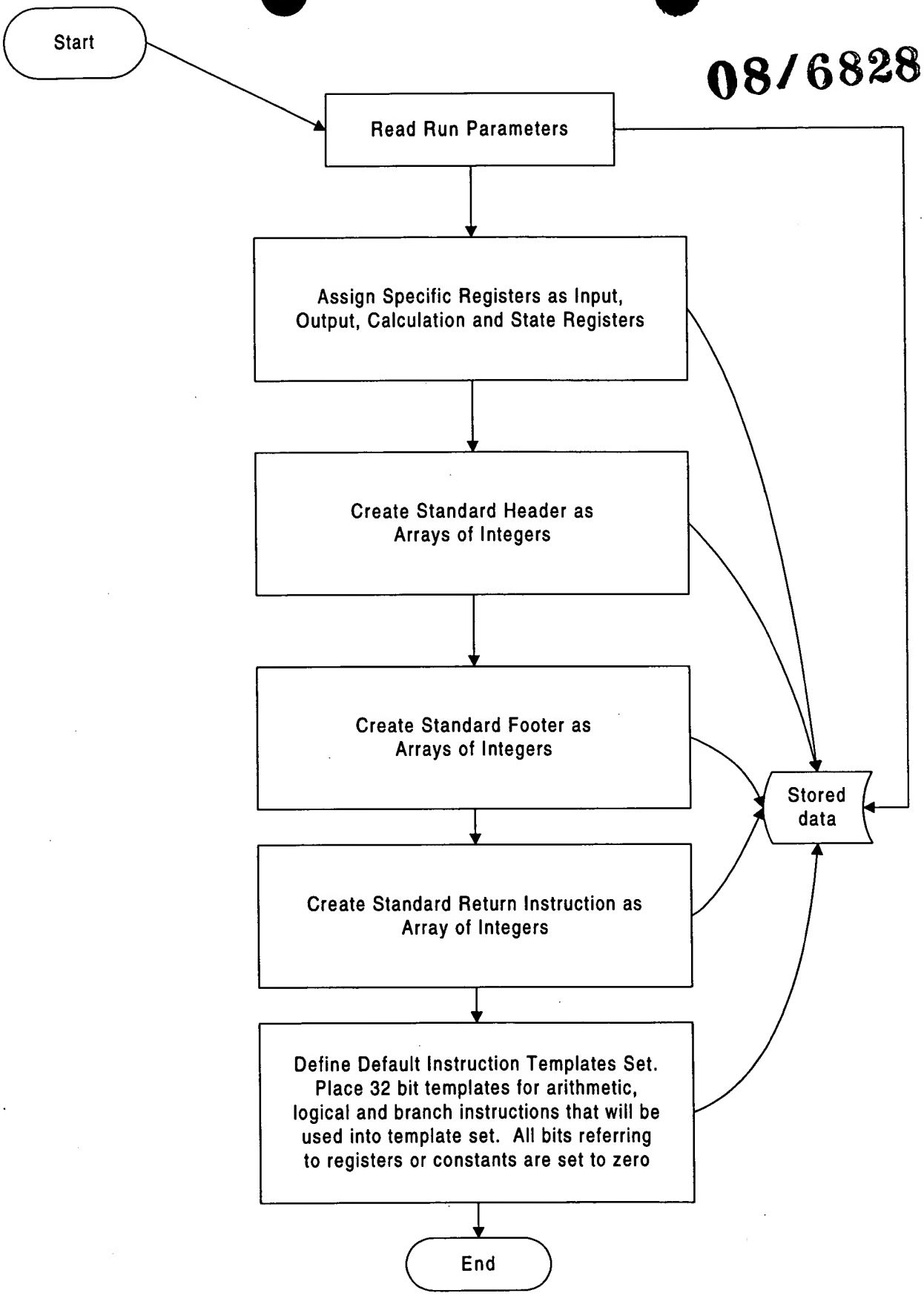


Figure 18b

Setup

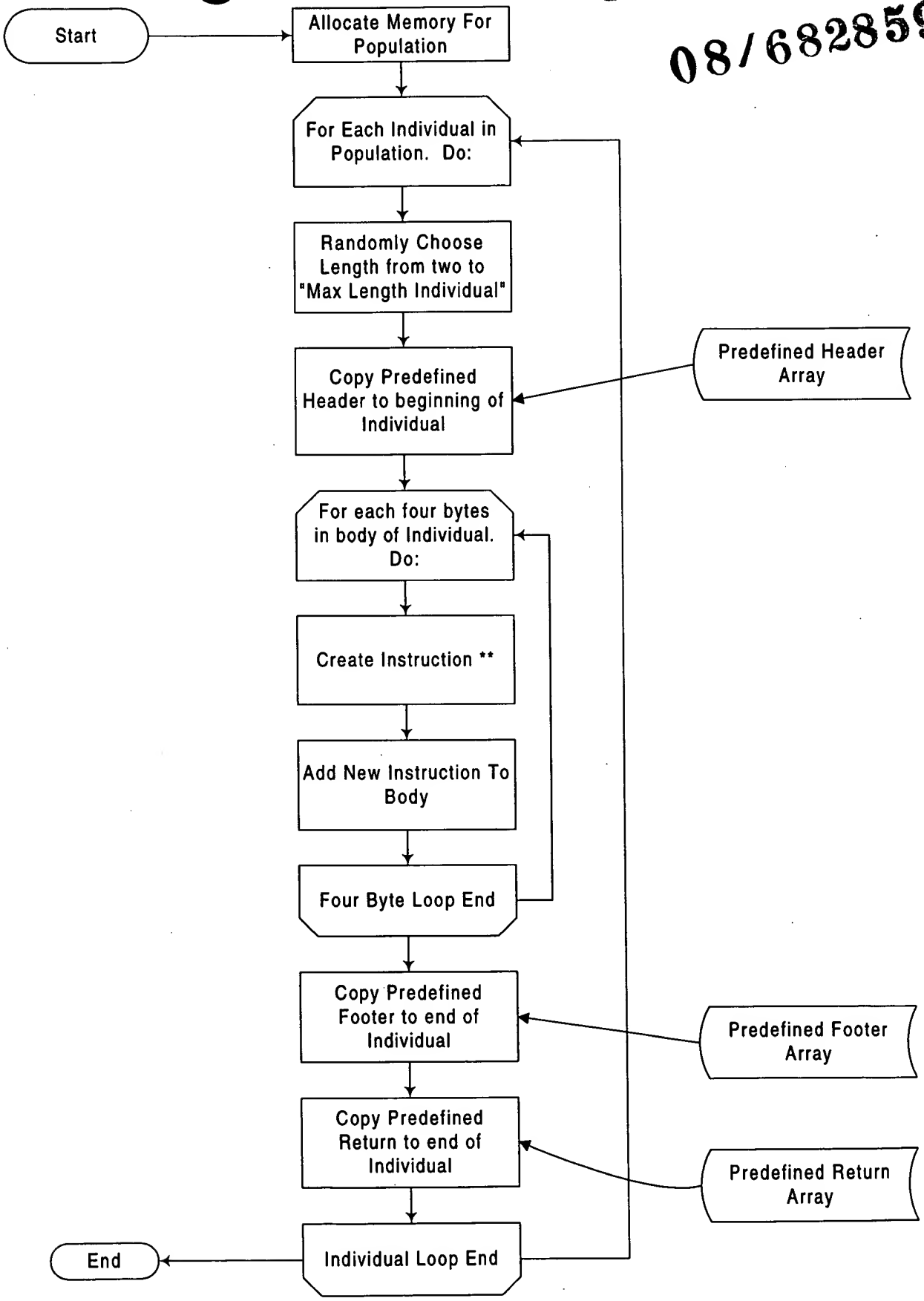


Figure 18c

Initialization

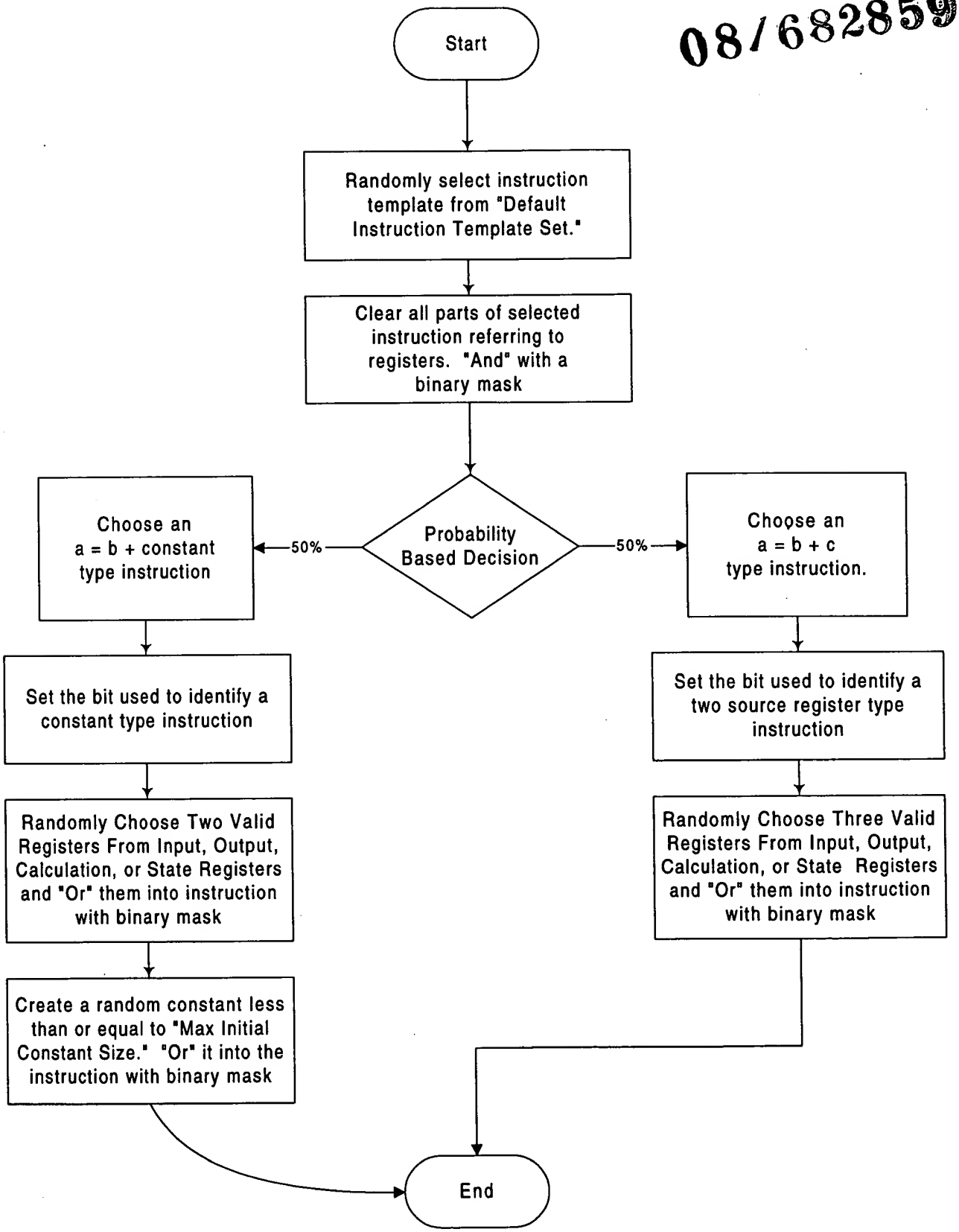


Figure 18d

Create Instruction

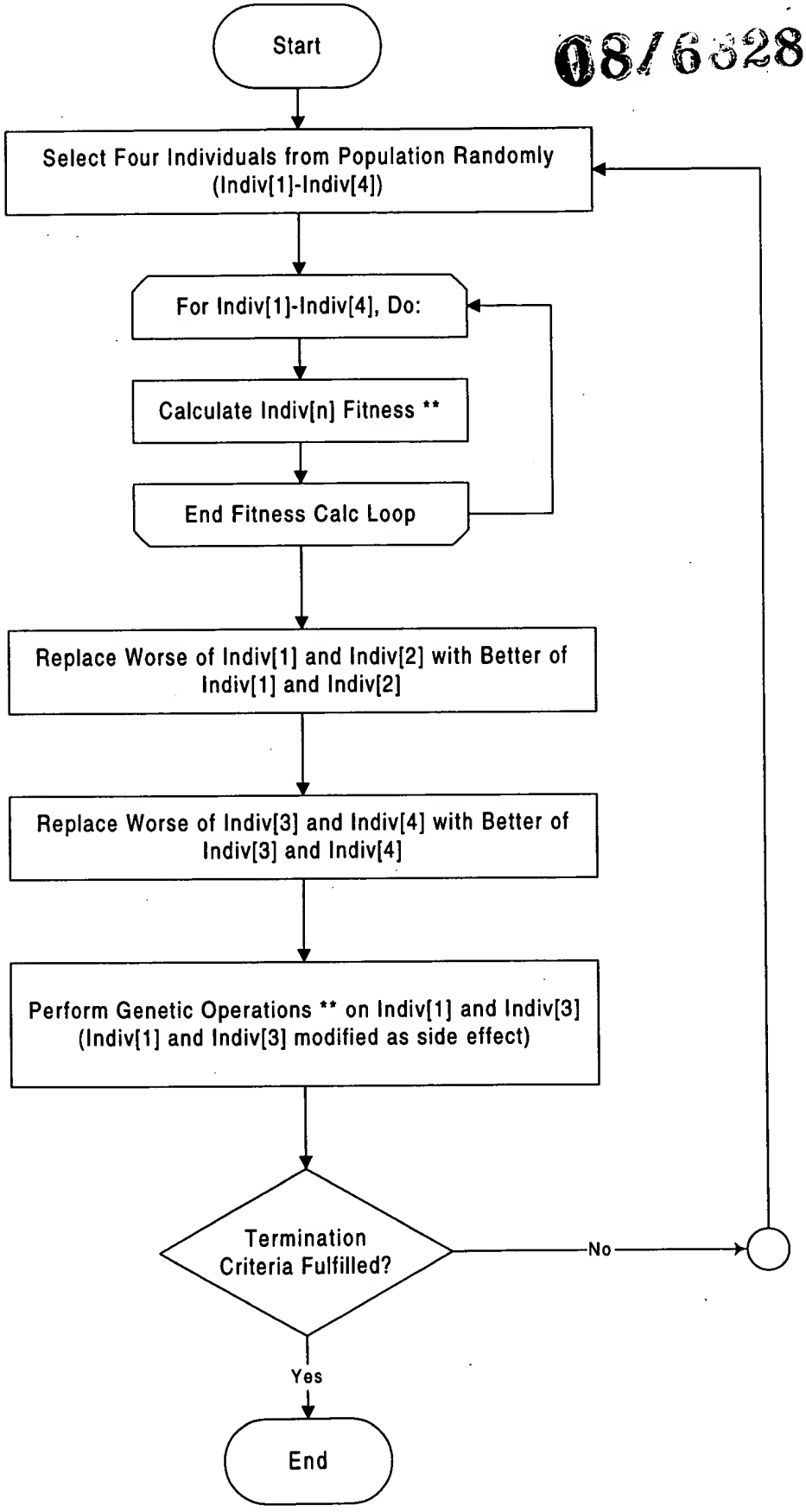
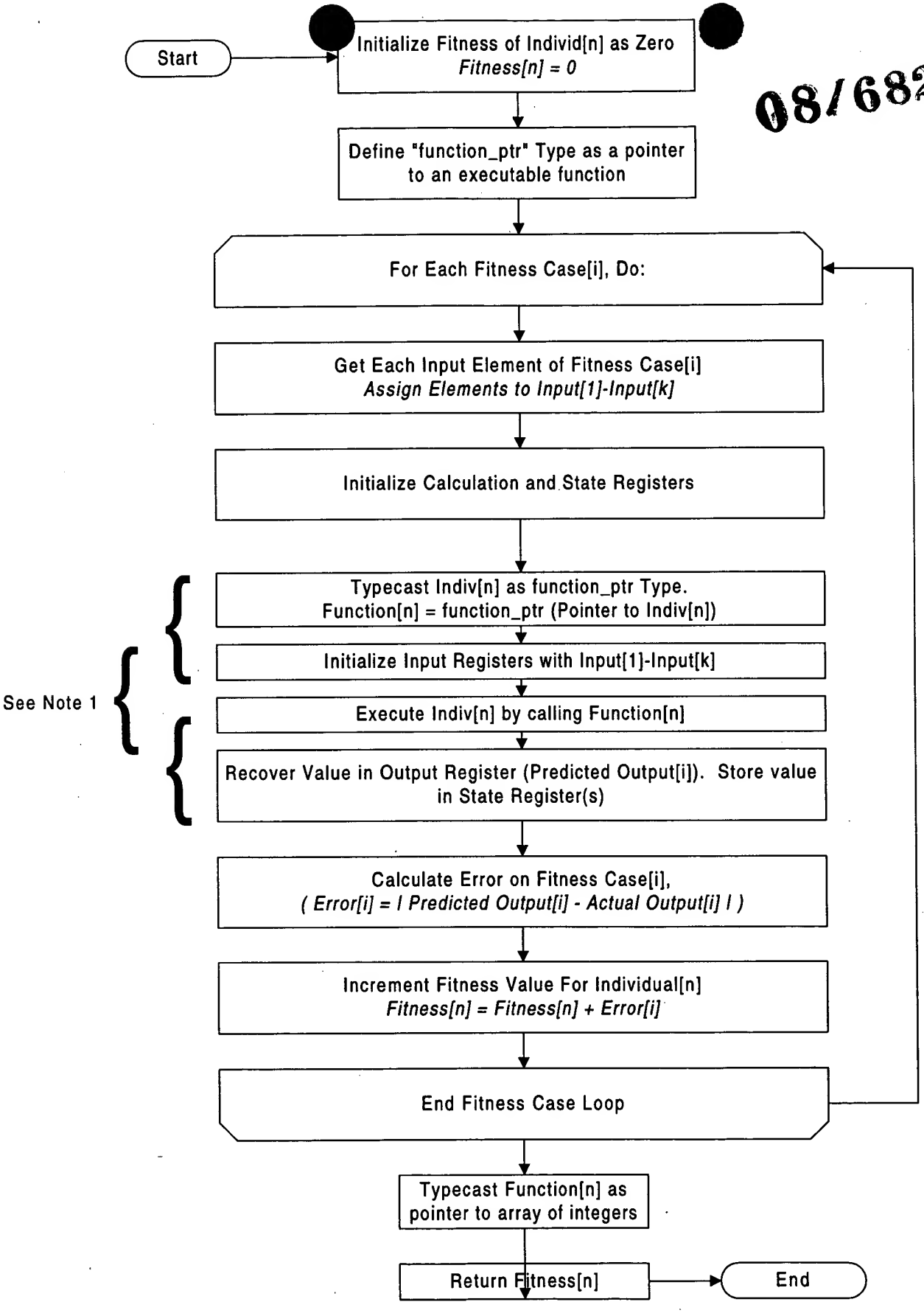


Figure 18e

Main CGPS Loop



Note 1: All of these steps are accomplished with the code:
Predicted_Output[i] = ((function_ptr) Indiv[n])(Input[1]. . .Input[k])

Figure 18f

Calculate Indiv[n] Fitness

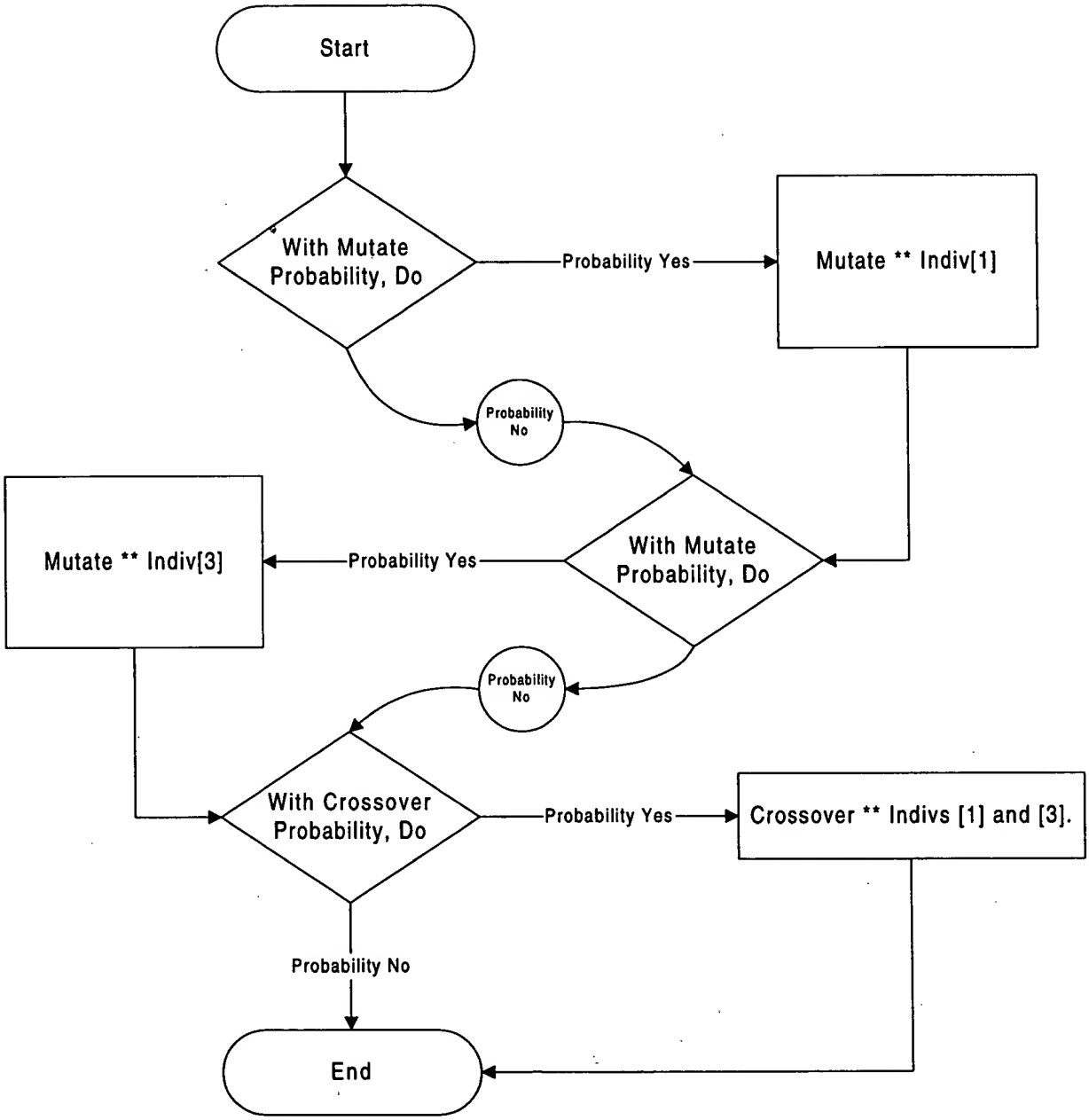


Figure 18g

Perform Genetic Operations

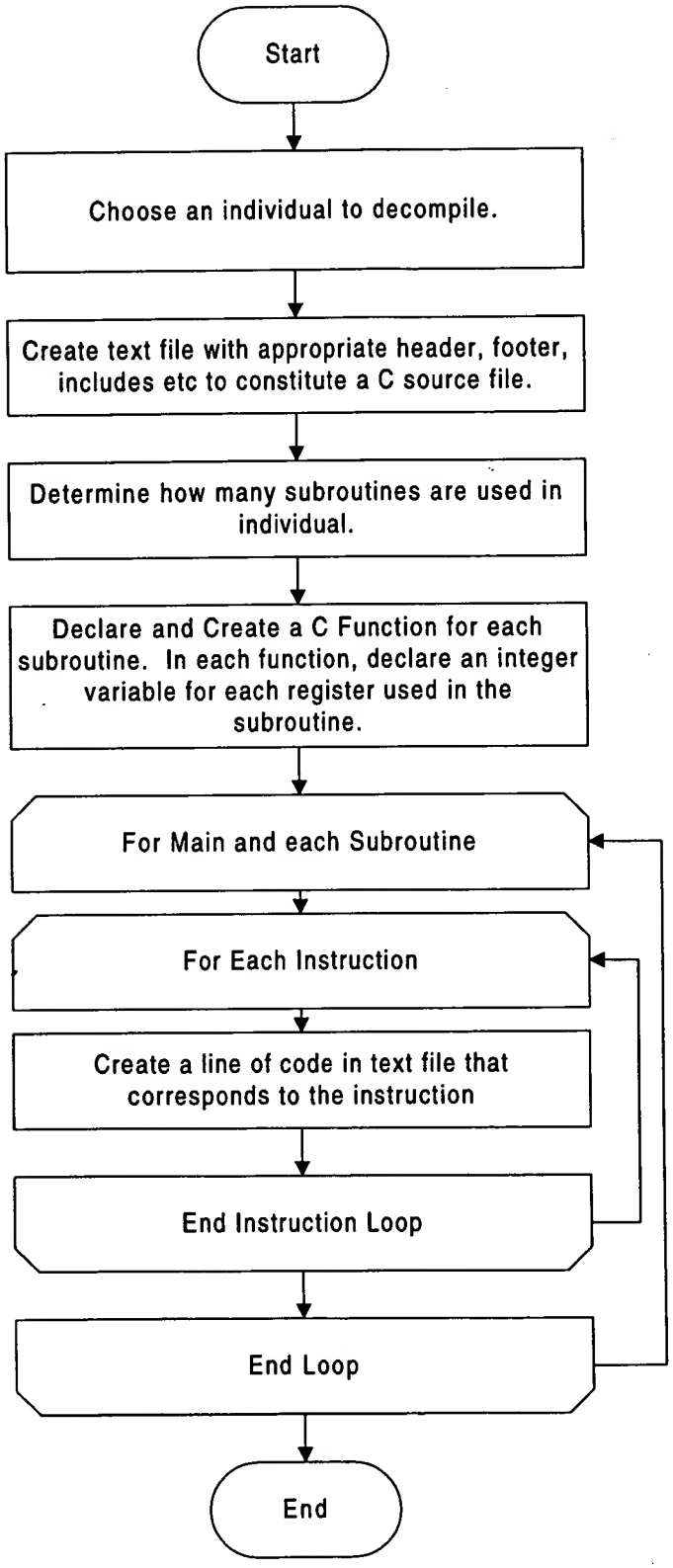

Start
Choose an individual to decompile.
Create text file with appropriate header, footer, includes etc to constitute a C source file.
Determine how many subroutines are used in individual.
Declare and Create a C Function for each subroutine. In each function, declare an integer variable for each register used in the subroutine.
For Main and each Subroutine
For Each Instruction
Create a line of code in text file that corresponds to the instruction
End Instruction Loop
End Loop
End

Figure 18h

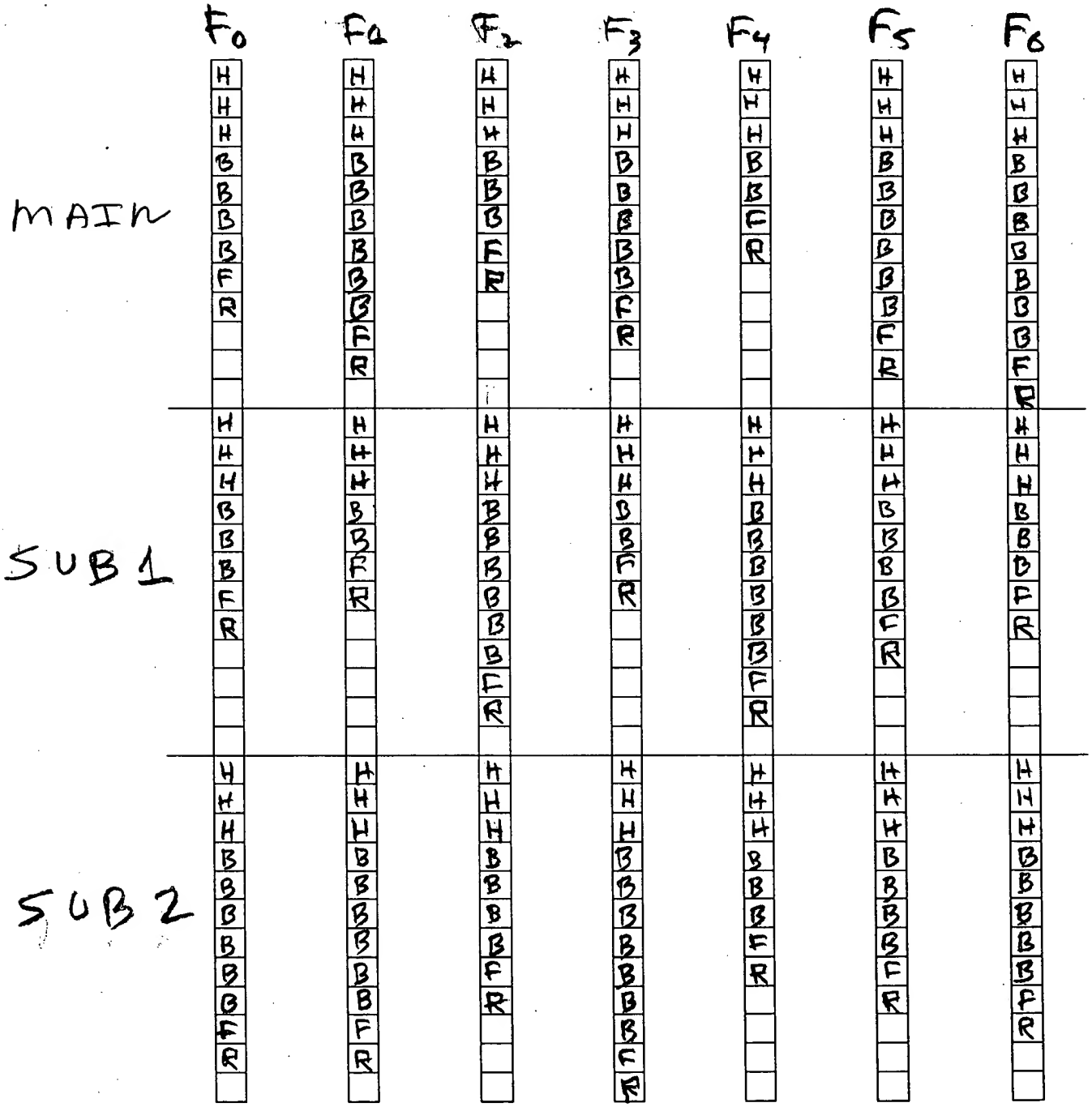


Fig. 19

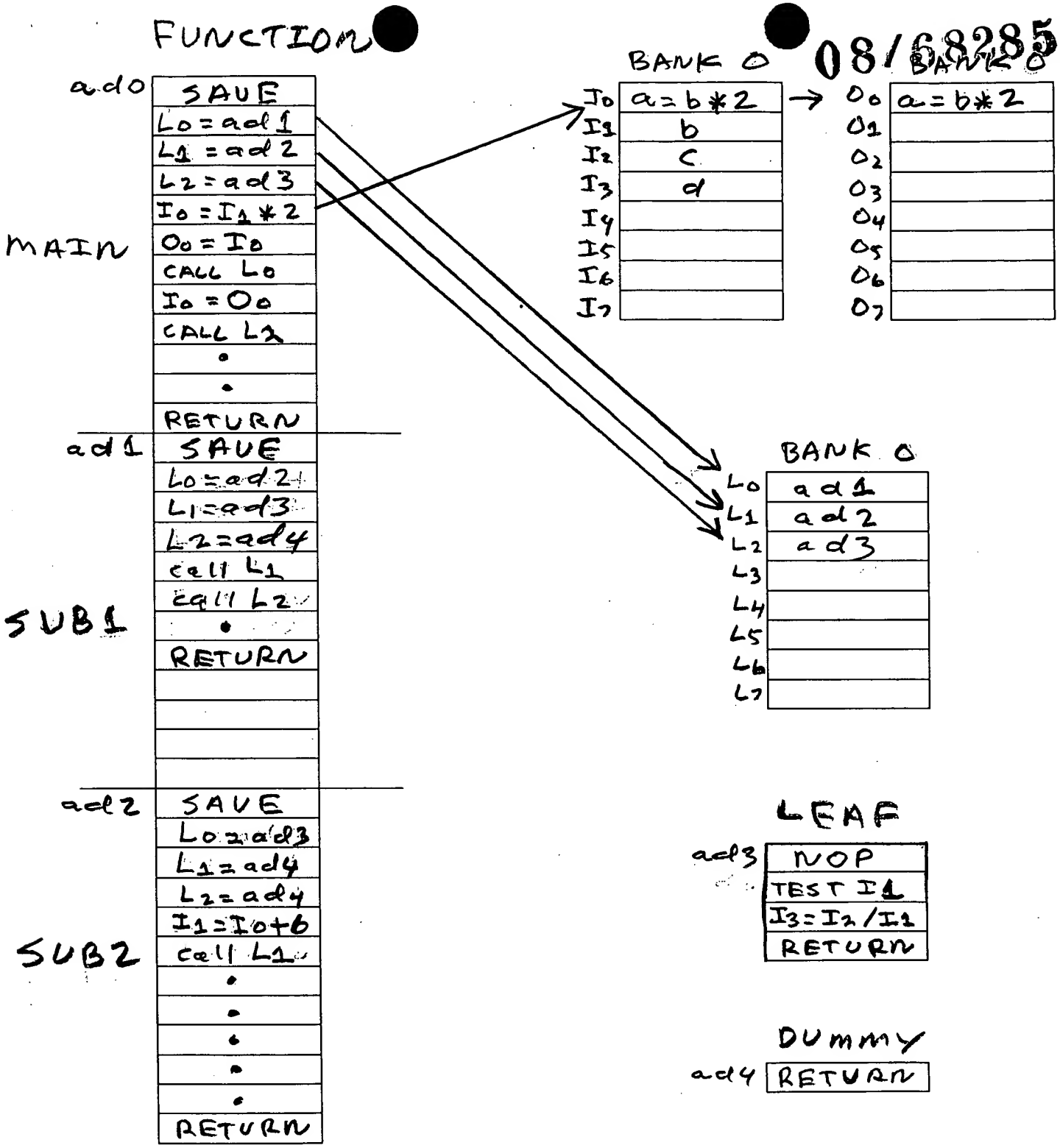

FUNCTION
add0
SAVE
L0 = add1
L1 = add2
L2 = add3
I0 = I1 * 2
MAIN
O0 = I0
CALL L0
I0 = O0
CALL L2
RETURN
add1
SAVE
L0 = add2
L1 = add3
L2 = add4
call L1
call L2
SUB1
RETURN
add2
SAVE
L0 = add3
L1 = add4
L2 = add4
I1 = I0 + 6
SUB2
call L1
RETURN
BANK 0
I0 a = b * 2
I1 b
I2 c
I3 d
I4
I5
I6
I7
BANK 0
O0 a = b * 2
O1
O2
O3
O4
O5
O6
O7
BANK 0
L0 add1
L1 add2
L2 add3
L3
L4
L5
L6
L7
LEAF
add3
NOP
TEST I1
I3 = I2 / I1
RETURN
DUMMY
add4
RETURN

Fig. 20

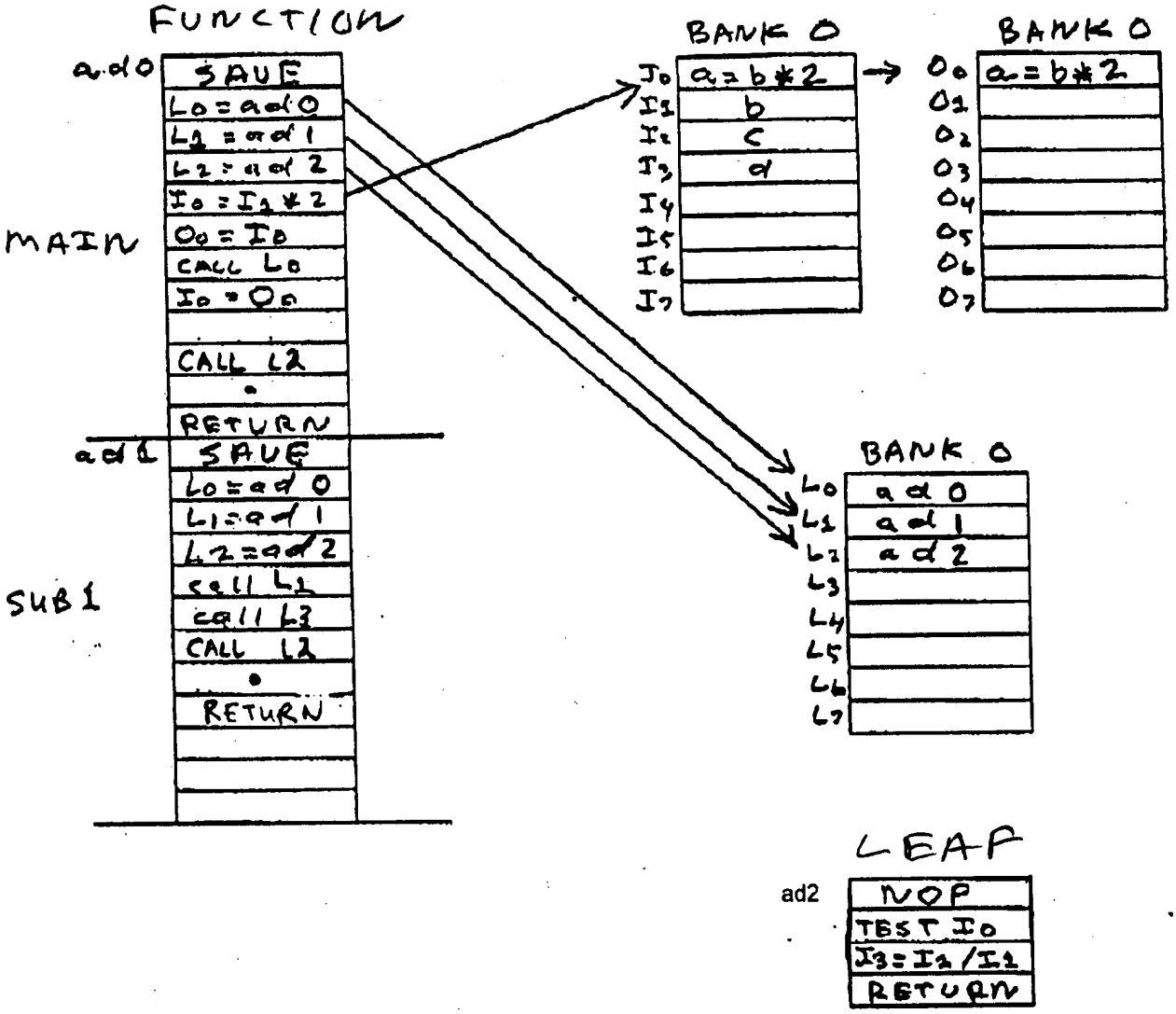


Fig. 21

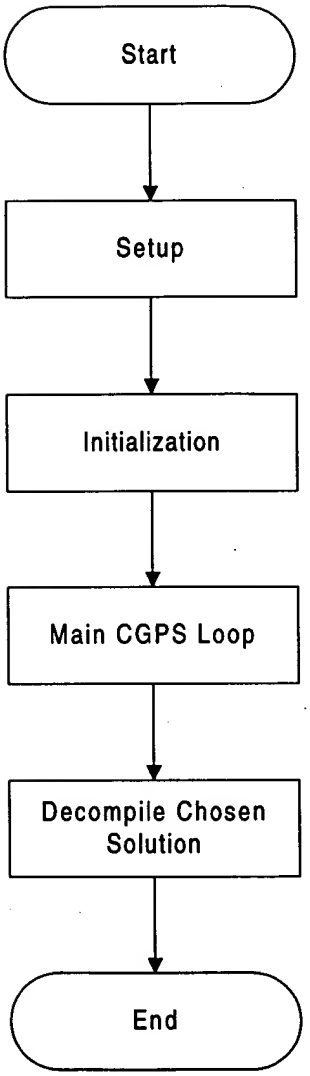

Start
Setup
Initialization
Main CGPS Loop
Decompile Chosen Solution
End

Figure 22a

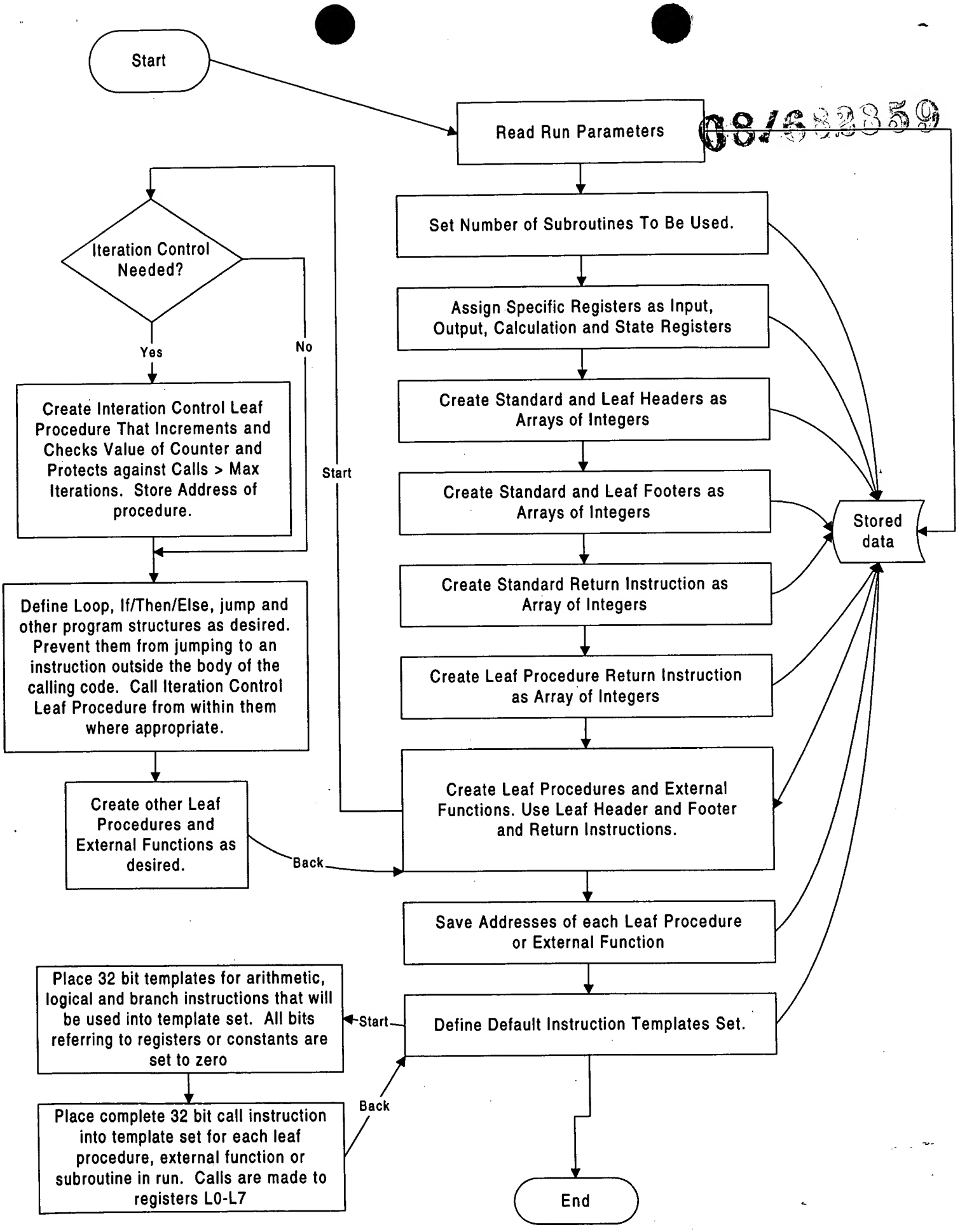

Start
Read Run Parameters
Set Number of Subroutines To Be Used.
Assign Specific Registers as Input, Output, Calculation and State Registers
Create Standard and Leaf Headers as Arrays of Integers
Create Standard and Leaf Footers as Arrays of Integers
Create Standard Return Instruction as Array of Integers
Create Leaf Procedure Return Instruction as Array of Integers
Create Leaf Procedures and External Functions. Use Leaf Header and Footer and Return Instructions.
Save Addresses of each Leaf Procedure or External Function
Define Default Instruction Templates Set.
Stored data
Iteration Control Needed?
Yes
No
Create Interation Control Leaf Procedure That Increments and Checks Value of Counter and Protects against Calls > Max Iterations. Store Address of procedure.
Define Loop, If/Then/Else, jump and other program structures as desired. Prevent them from jumping to an instruction outside the body of the calling code. Call Iteration Control Leaf Procedure from within them where appropriate.
Create other Leaf Procedures and External Functions as desired.
Start
Back
Place 32 bit templates for arithmetic, logical and branch instructions that will be used into template set. All bits referring to registers or constants are set to zero
Start
Place complete 32 bit call instruction into template set for each leaf procedure, external function or subroutine in run. Calls are made to registers L0-L7
Back
End

Figure 22b

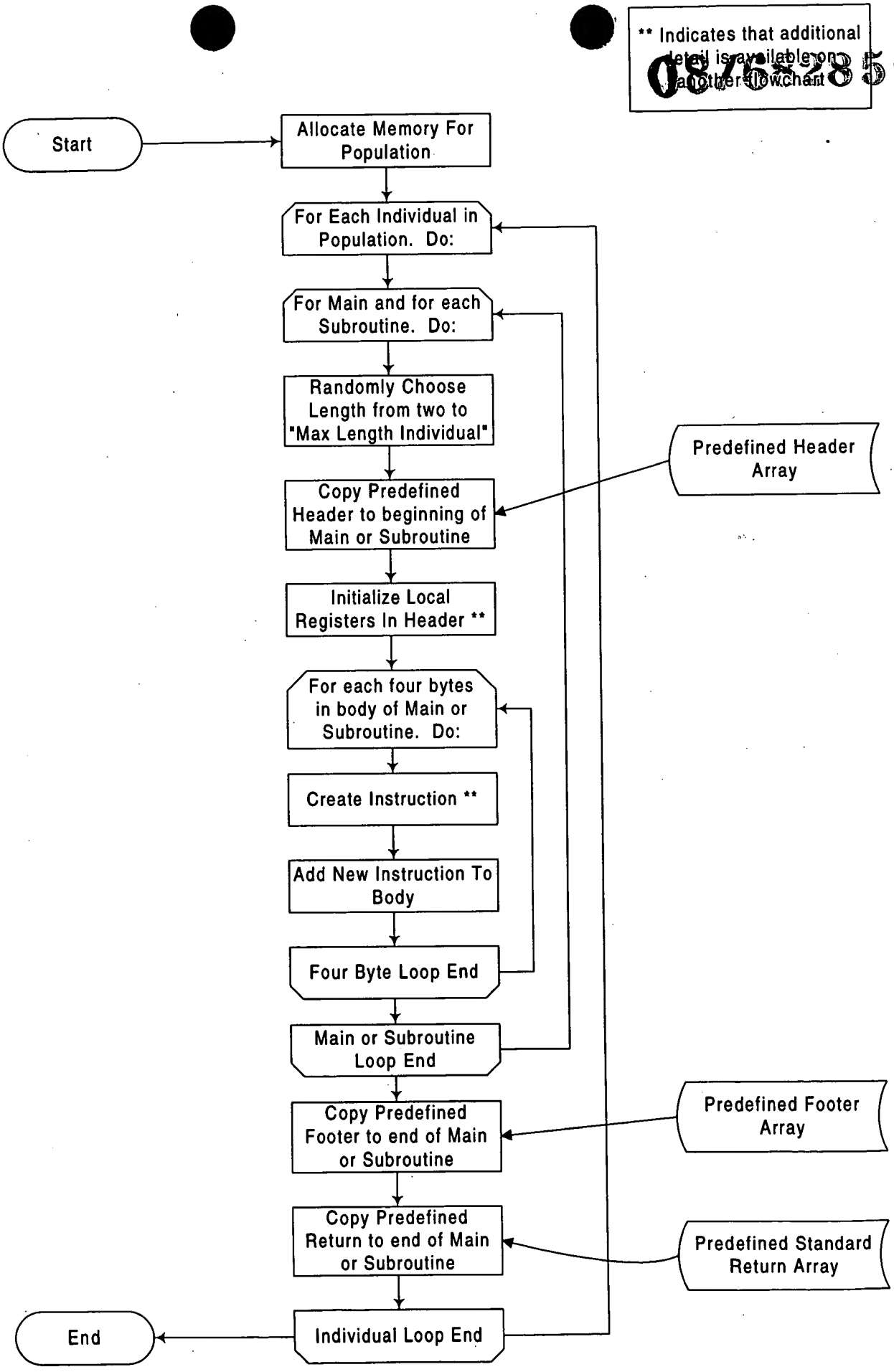

** Indicates that additional detail is available on another flowchart
Start
Allocate Memory For Population
For Each Individual in Population. Do:
For Main and for each Subroutine. Do:
Randomly Choose Length from two to "Max Length Individual"
Predefined Header Array
Copy Predefined Header to beginning of Main or Subroutine
Initialize Local Registers In Header **
For each four bytes in body of Main or Subroutine. Do:
Create Instruction **
Add New Instruction To Body
Four Byte Loop End
Main or Subroutine Loop End
Predefined Footer Array
Copy Predefined Footer to end of Main or Subroutine
Copy Predefined Return to end of Main or Subroutine
Predefined Standard Return Array
End
Individual Loop End

Figure 22c

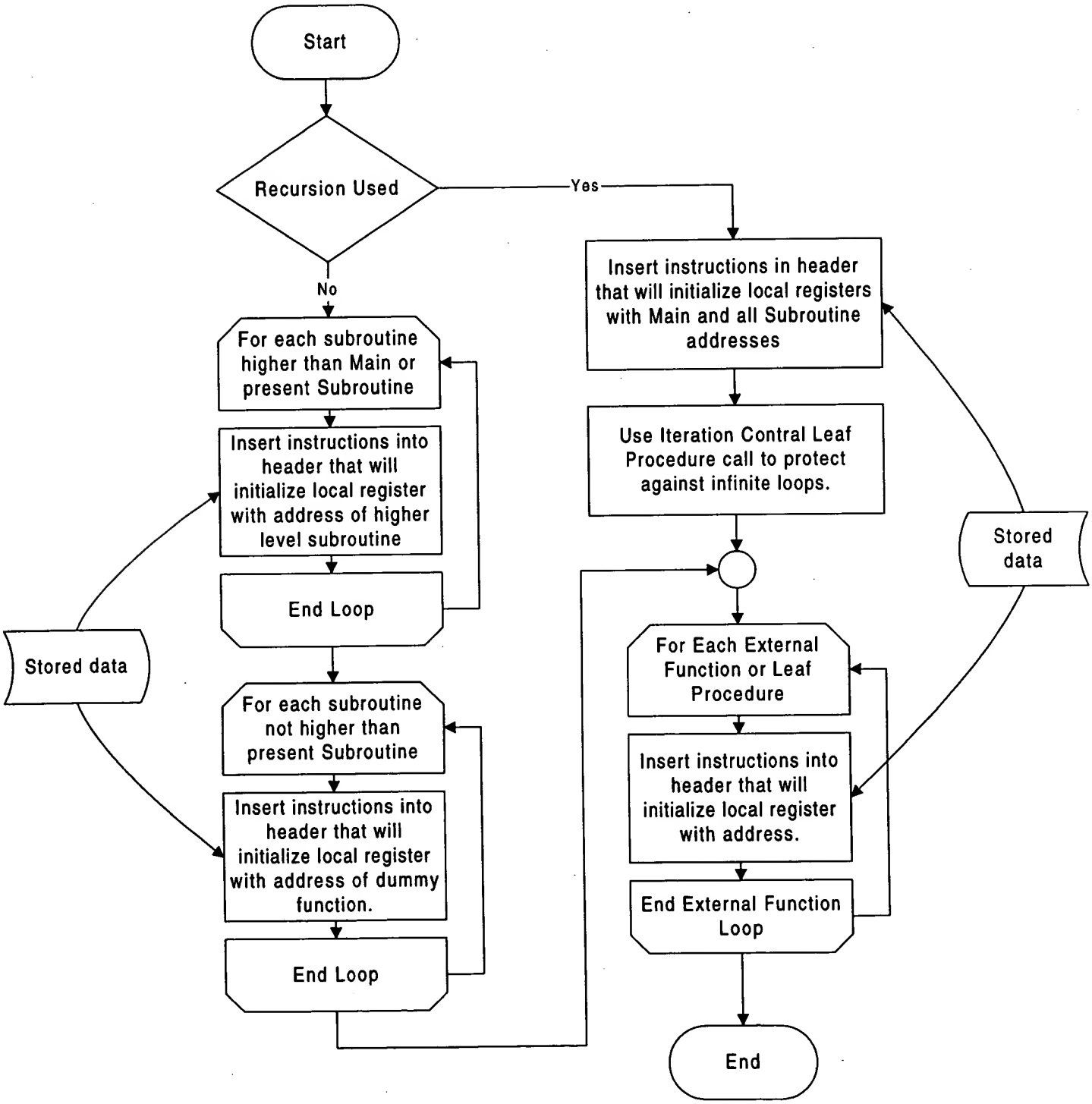

Start
Recursion Used
Yes
No
Insert instructions in header that will initialize local registers with Main and all Subroutine addresses
Use Iteration Contral Leaf Procedure call to protect against infinite loops.
Stored data
For each subroutine higher than Main or present Subroutine
Insert instructions into header that will initialize local register with address of higher level subroutine
End Loop
Stored data
For each subroutine not higher than present Subroutine
Insert instructions into header that will initialize local register with address of dummy function.
End Loop
For Each External Function or Leaf Procedure
Insert instructions into header that will initialize local register with address.
End External Function Loop
End

Figure 22d

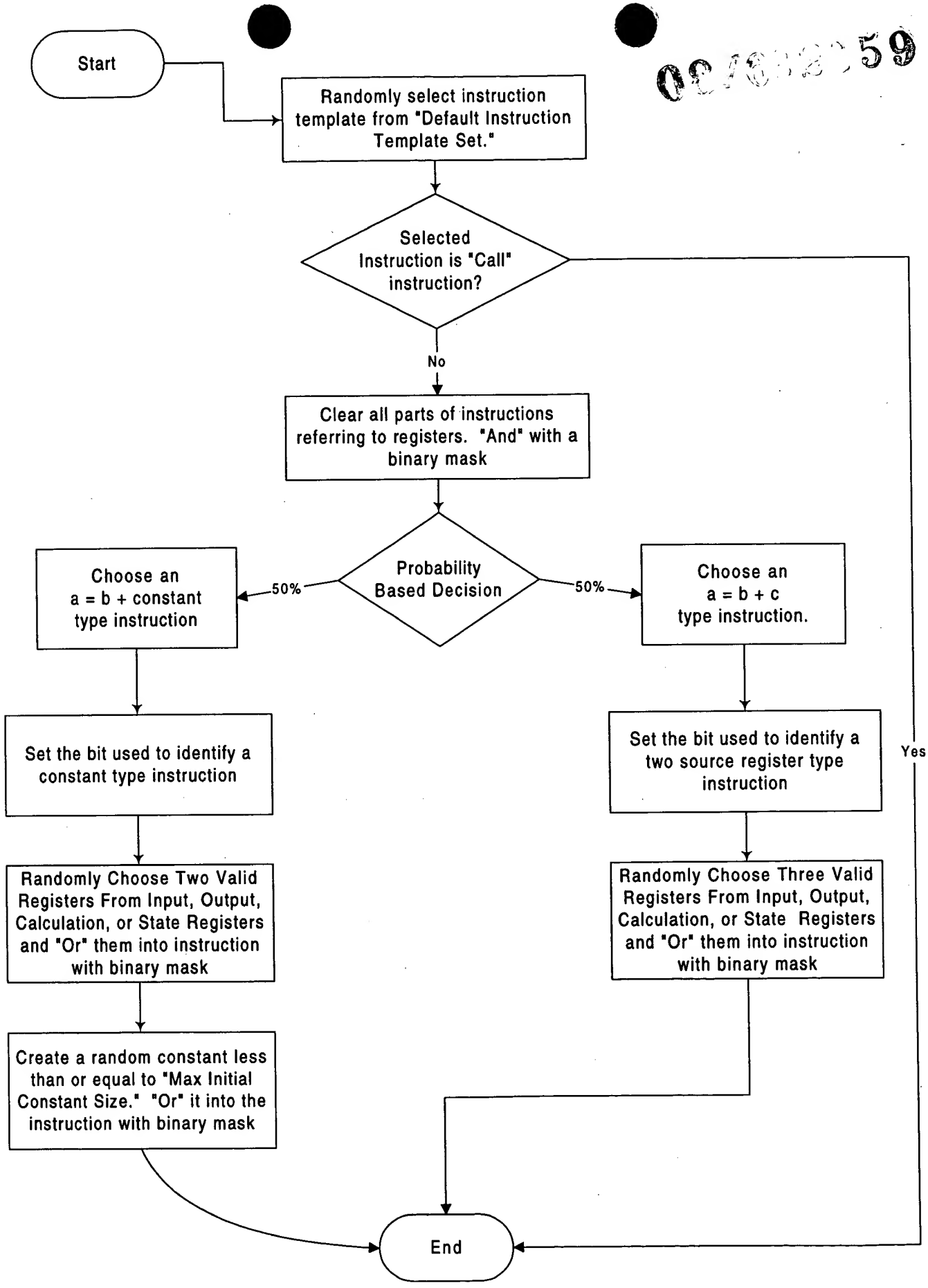

Start
Randomly select instruction template from "Default Instruction Template Set."
Selected Instruction is "Call" instruction?
Yes
No
Clear all parts of instructions referring to registers. "And" with a binary mask
Probability Based Decision
50%
50%
Choose an a = b + constant type instruction
Choose an a = b + c type instruction.
Set the bit used to identify a constant type instruction
Set the bit used to identify a two source register type instruction
Randomly Choose Two Valid Registers From Input, Output, Calculation, or State Registers and "Or" them into instruction with binary mask
Randomly Choose Three Valid Registers From Input, Output, Calculation, or State Registers and "Or" them into instruction with binary mask
Create a random constant less than or equal to "Max Initial Constant Size." "Or" it into the instruction with binary mask
End

Figure 22e

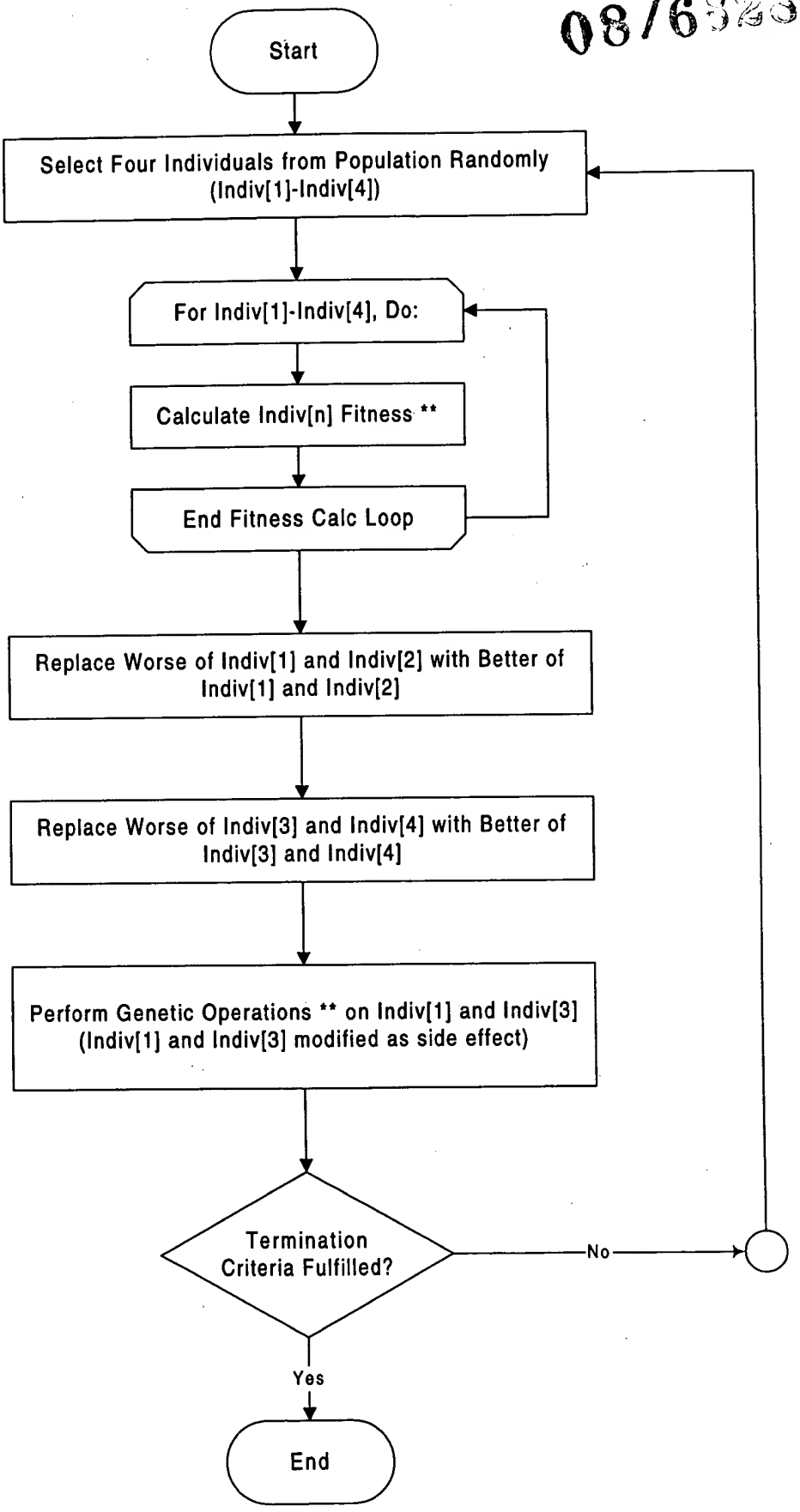

Start
Select Four Individuals from Population Randomly (Indiv[1]-Indiv[4])
For Indiv[1]-Indiv[4], Do:
Calculate Indiv[n] Fitness **
End Fitness Calc Loop
Replace Worse of Indiv[1] and Indiv[2] with Better of Indiv[1] and Indiv[2]
Replace Worse of Indiv[3] and Indiv[4] with Better of Indiv[3] and Indiv[4]
Perform Genetic Operations ** on Indiv[1] and Indiv[3] (Indiv[1] and Indiv[3] modified as side effect)
Termination Criteria Fulfilled?
No
Yes
End

Figure 22f

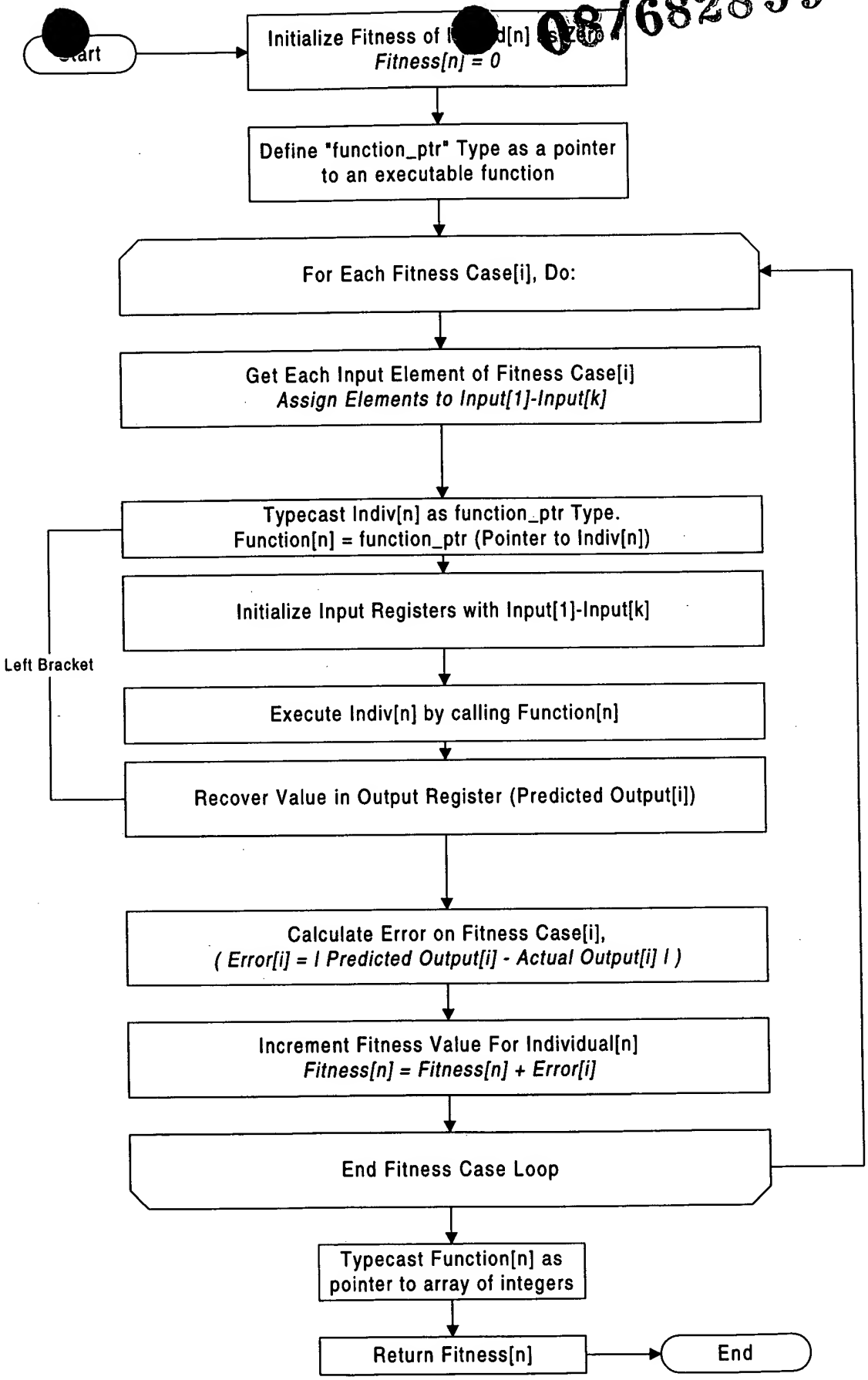


Figure 22g

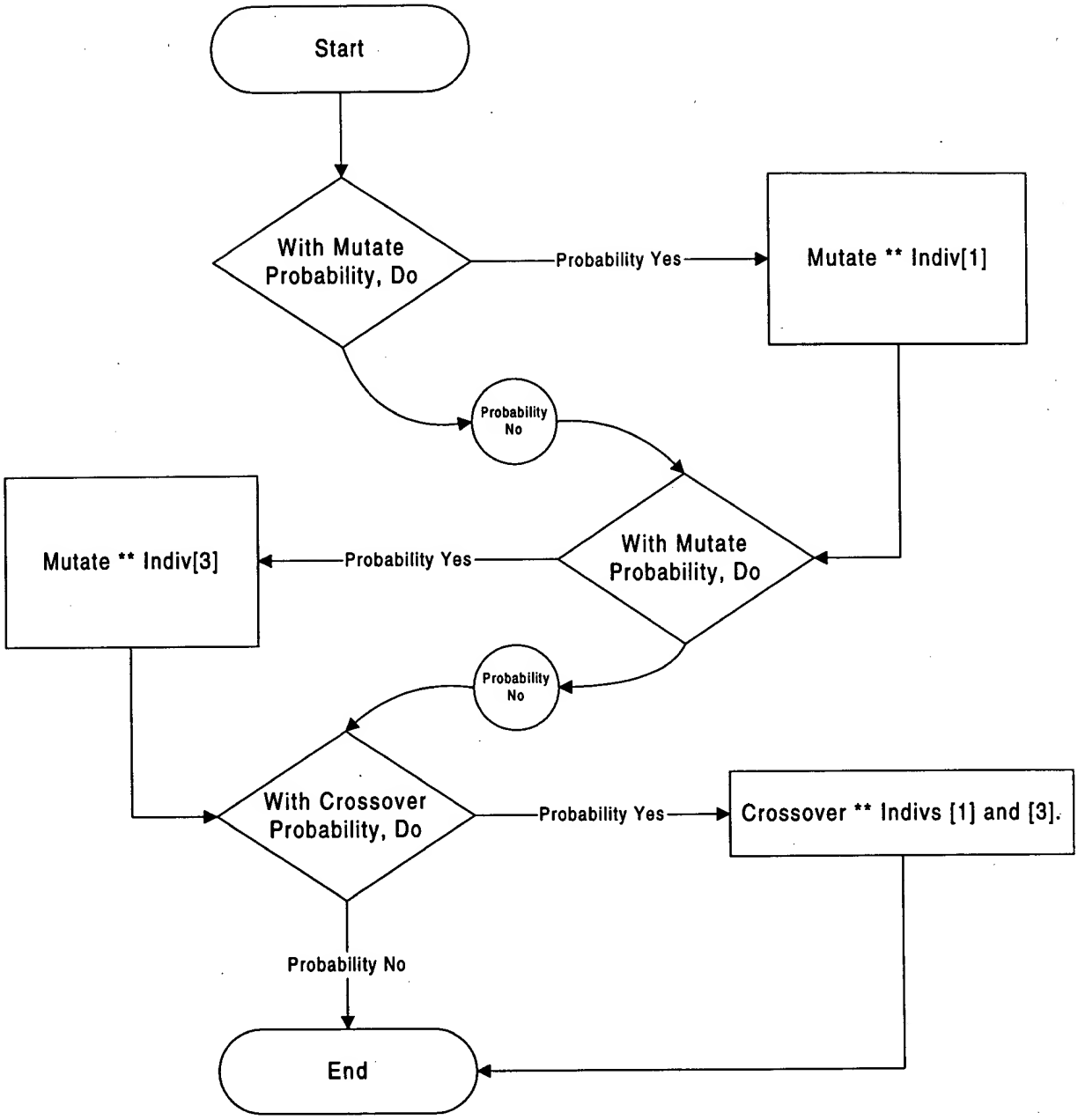

Start
With Mutate Probability, Do
Probability Yes
Mutate ** Indiv[1]
Probability No
With Mutate Probability, Do
Probability Yes
Mutate ** Indiv[3]
Probability No
With Crossover Probability, Do
Probability Yes
Crossover ** Indivs [1] and [3].
Probability No
End

Figure 22h

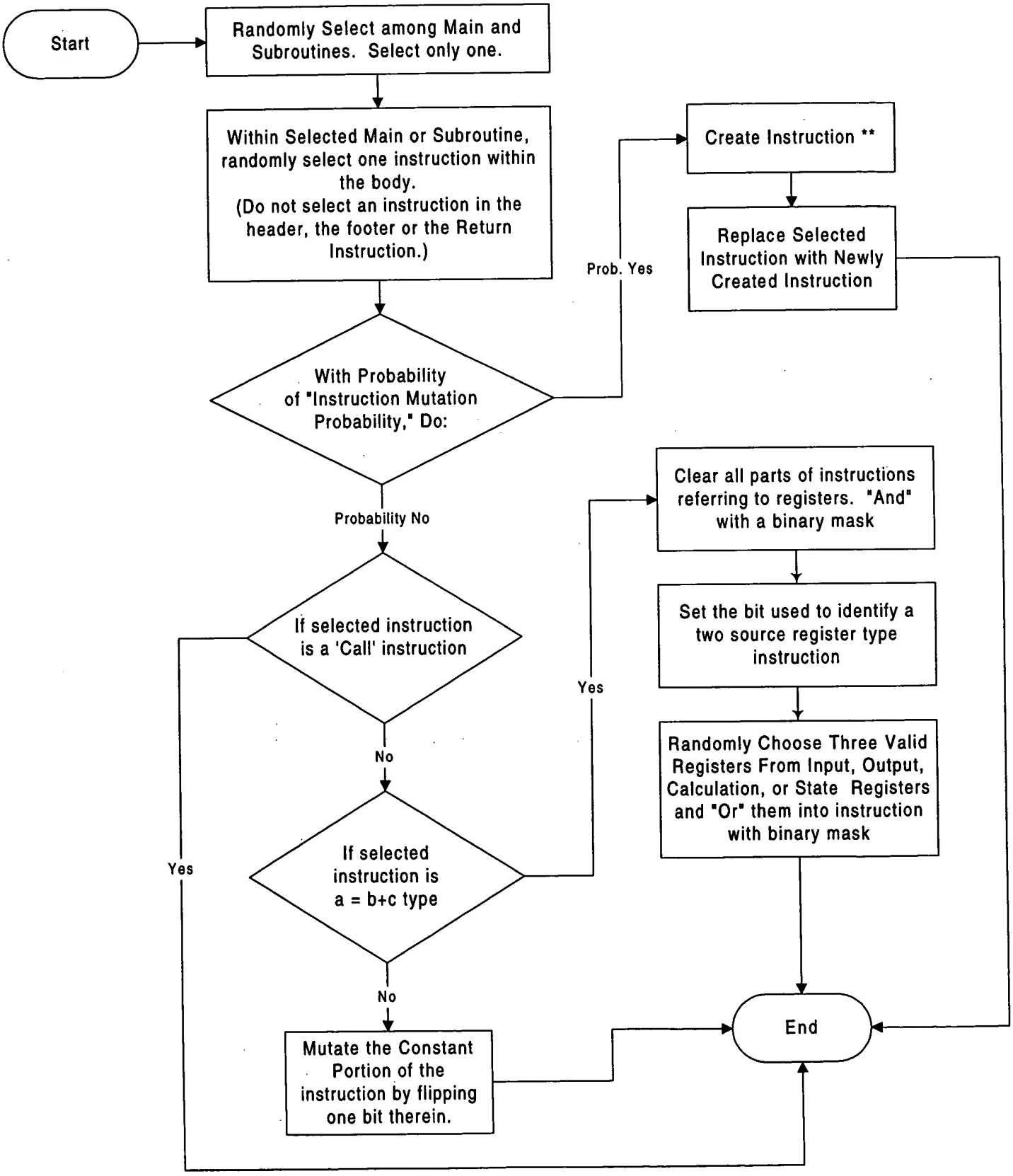

Start
Randomly Select among Main and Subroutines. Select only one.
Within Selected Main or Subroutine, randomly select one instruction within the body. (Do not select an instruction in the header, the footer or the Return Instruction.)
With Probability of "Instruction Mutation Probability," Do:
Prob. Yes
Create Instruction **
Replace Selected Instruction with Newly Created Instruction
Probability No
If selected instruction is a 'Call' instruction
Yes
No
If selected instruction is a = b+c type
Yes
No
Clear all parts of instructions referring to registers. "And" with a binary mask
Set the bit used to identify a two source register type instruction
Randomly Choose Three Valid Registers From Input, Output, Calculation, or State Registers and "Or" them into instruction with binary mask
Mutate the Constant Portion of the instruction by flipping one bit therein.
End

Figure 22i

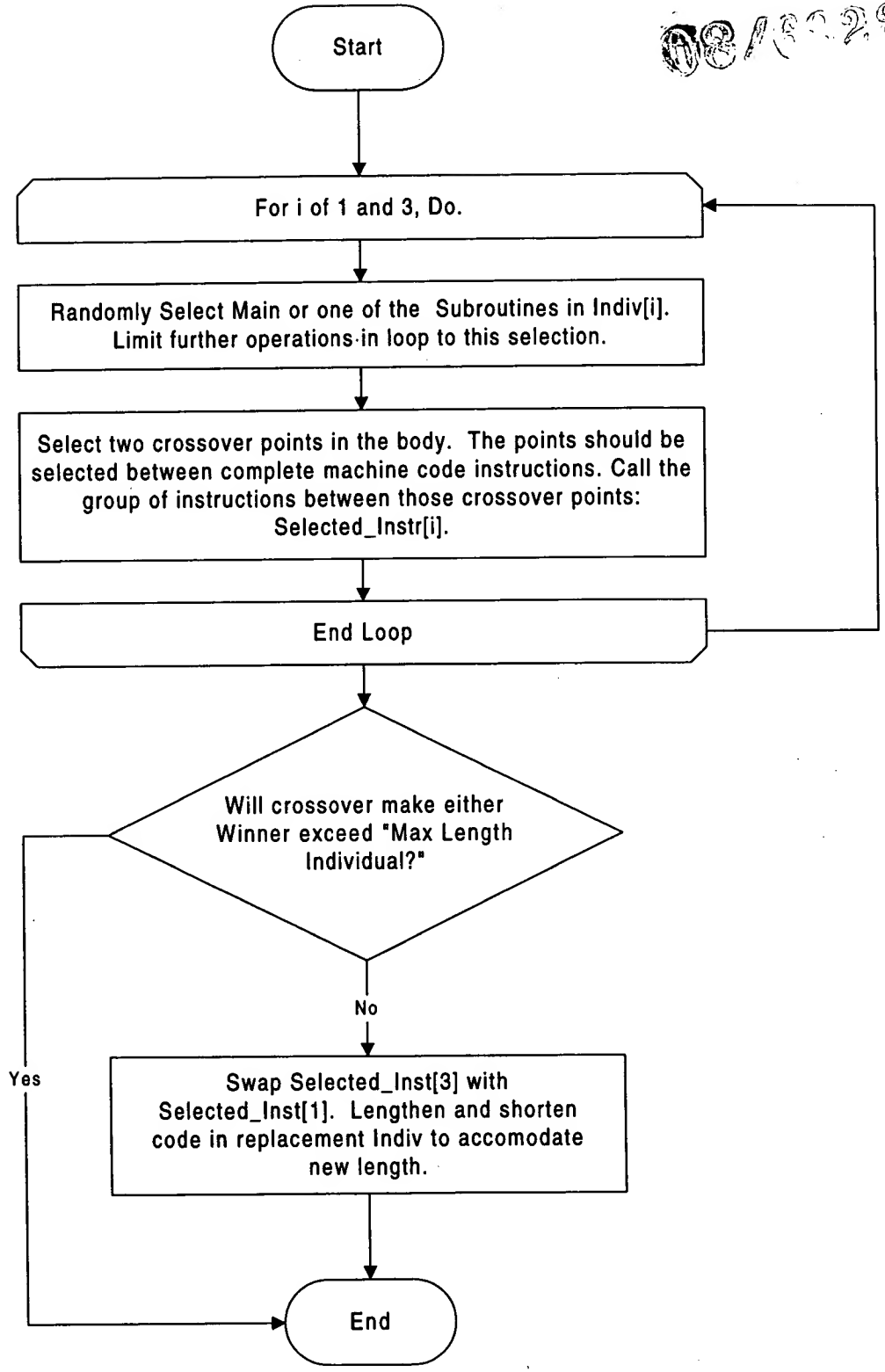


Figure 22j

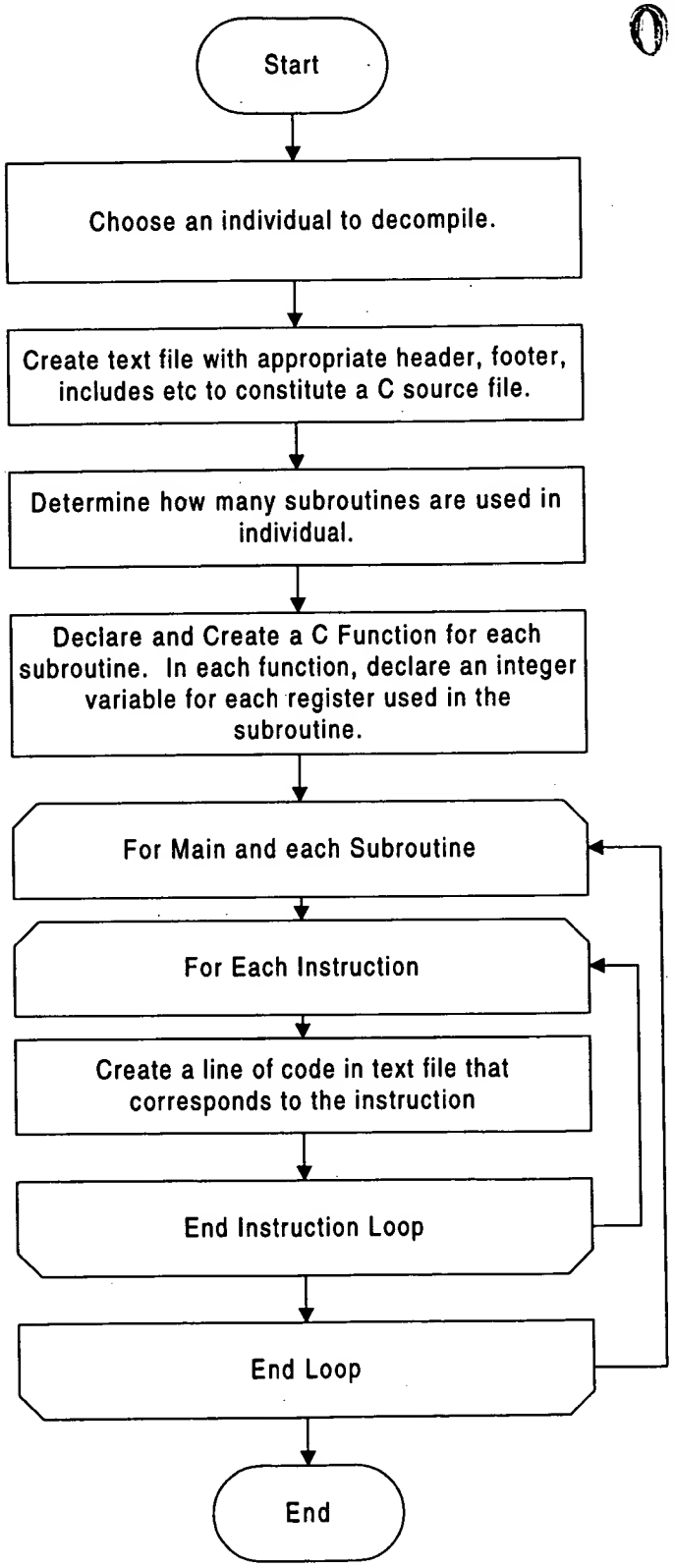

Start
Choose an individual to decompile.
Create text file with appropriate header, footer, includes etc to constitute a C source file.
Determine how many subroutines are used in individual.
Declare and Create a C Function for each subroutine. In each function, declare an integer variable for each register used in the subroutine.
For Main and each Subroutine
For Each Instruction
Create a line of code in text file that corresponds to the instruction
End Instruction Loop
End Loop
End

Figure 22k